



Performance with Open/SpeedShop



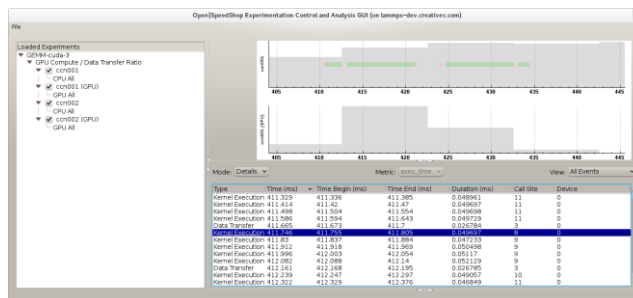
NASA Open/SpeedShop Update/Training

Jim Galarowicz: Argo Navis, Krell Institute

Greg Schultz, Argo Navis

Don Maghrak, Krell Institute


William Hachfeld, Argo Navis, Krell Institute





Webinar Preparation Underway

If you cannot hear meeting room activity on your computer:

- 1. Be sure your computer audio volume is high enough to hear*
- 2. If there's still a problem, use the "Chat Box" facility of Webex to request a telephone callback. (The Chat box is reachable from the green tab  at the top of your screen.)*

How to hear the audio (in order of preference):

1. Use your computer audio to follow the training
 - This should work with Windows and Mac, but maybe not with Linux
2. Dial 650-479-3208

Participants not in the meeting room at NAS have been muted

- This should reduce cross-talk and provide everyone with better audio

Questions are welcome during the presentation:

- Ask a question in the Chat facility
 - The host will be monitoring and will relay your question to the speaker
- If the question is too complicated to ask in text, call 650-479-3208 and ask the host to unmute you in order for you to communicate by phone.

- ❖ **Jim Galarowicz, Argo Navis, Krell Institute**
- ❖ **Greg Schultz, Argo Navis**
- ❖ **Don Maghrak, Krell Institute**
- ❖ **William Hachfeld, Argo Navis, Krell Institute**



Open | SpeedShop extended team:

- ❖ **Patrick Romero: Krell Institute**
- ❖ **Jennifer Green, David Montoya, Mike Mason, David Shrader: LANL**
- ❖ **Martin Schulz, Matt Legendre and Chris Chambreau: LLNL**
- ❖ **Mahesh Rajan, Doug Pase, Anthony Agelastos: SNL**
- ❖ **Dyninst group (Bart Miller: UW & Jeff Hollingsworth: UMD)**
- ❖ **Phil Roth, Mike Brim: ORNL**
- ❖ **Ciera Jaspan: CMU**



Section 1: Introduction to Open | SpeedShop tools

- How to use Open | SpeedShop to gather and display
- Overview of performance experiments
 - Sampling Experiments and Tracing Experiments
- How to compare performance data for different application runs

Section 2: New Functionality/Experiments

- Memory (ossmem) experiment
- OpenMP augmentation and OMPTP (ossomptp) experiment
- POSIX threads (osspthread) experiment
- Lightweight experiments (ossiop, ossmpip)
- NVIDIA CUDA tracing experiment (osscuda)

Section 3: Roadmap / Future Plans

Supplemental Information

- Command Line Interface (CLI) tutorial and examples

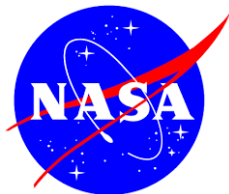
Pleades platform:

- ❖ module use /home4/jgalarow/privatemodules
- ❖ Module names:
 - module load openspeedshop (defaults to mpt)
 - module load openspeedshop.mpt
 - module load openspeedshop.intelmpi
 - module load openspeedshop.mvapich2
 - module load openspeedshop.openmpi

KNL cluster platform:

- ❖ module use /u/jgalarow/privatemodules
- ❖ Module names:
 - module load openspeedshop (defaults to mpt)
 - module load openspeedshop.mpt
 - module load openspeedshop.intelmpi

For mpi* experiments use the module file that corresponds to the MPI implementation your application was built with.



Open | SpeedShop™

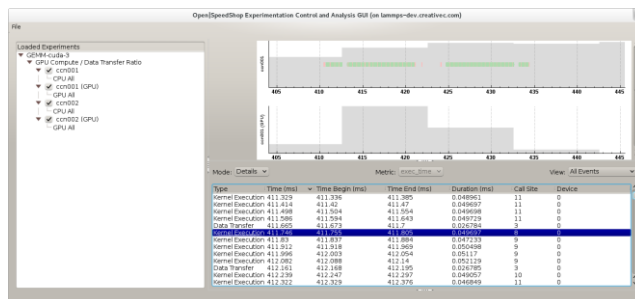
Performance with Open/SpeedShop



NASA Open/SpeedShop Update/Training

Section 1

Introduction into Tools and Open/SpeedShop



❖ Open Source Performance Analysis Tool Framework

- Most common performance analysis steps ***all in one tool***
- Combines ***tracing*** and ***sampling*** techniques
- ***No need to recompile the application being monitored.***
- Gathers and displays several types of performance information
- Maps performance data information to application source code

❖ Flexible and Easy to use

- User access through:
GUI, Command Line, Python Scripting, convenience scripts

❖ Scalable Data Collection

- Instrumentation of ***unmodified application binaries***
- New option for ***hierarchical online data aggregation***

❖ Supports a wide range of systems

- Extensively used and tested on a variety of ***Linux clusters***
- ***Cray, Blue Gene, ARM, Intel MIC, GPU*** support

Open|SpeedShop Workflow - default

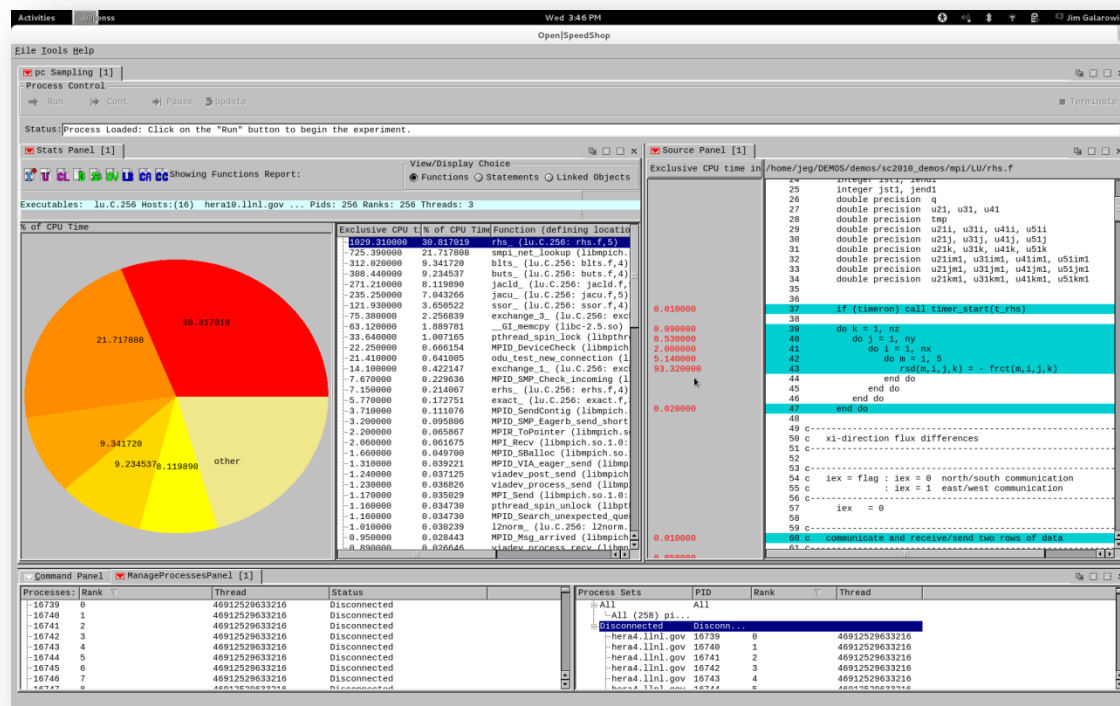
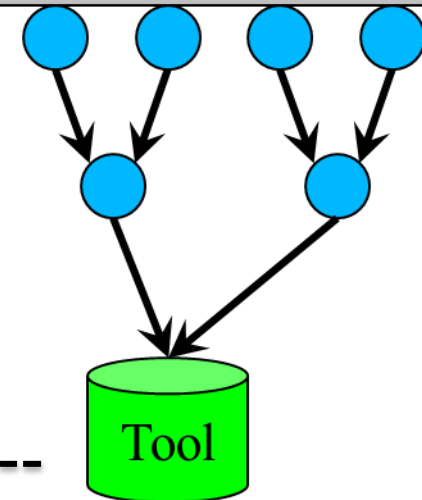


```
mpiexec_mpt -np 4 smg2000 -n 65 65 65
```

```
osspsamp "mpiexec_mpt -np 4 smg2000 -n 65 65 65"
```



MPI Application



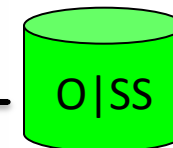
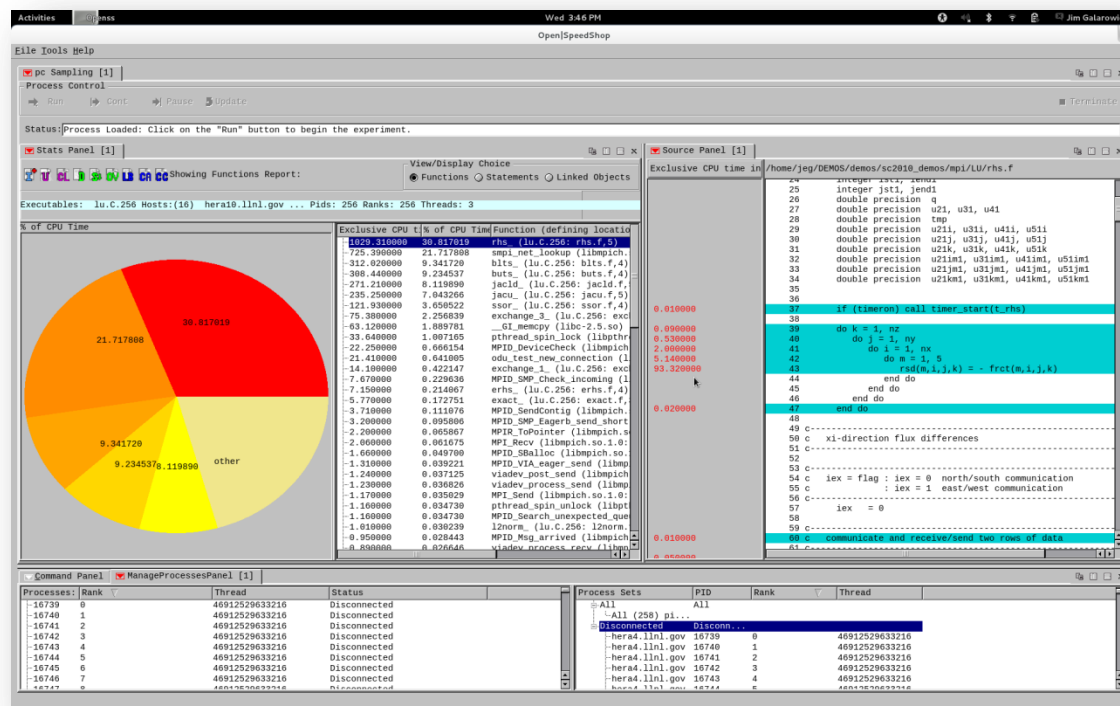
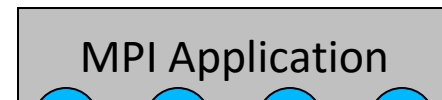
<http://www.openspeedshop.org/>

Open | SpeedShop Workflow – optional



```
mpiexec_mpt -np 4 smg2000 -n 65 65 65
```

```
ospspcsamp --offline "mpiexec_mpt -np 4 smg2000 -n 65 65 65"
```



Post-mortem

<http://www.openspeedshop.org/>

❖ Scripting language

- Immediate command interface
- O|SS interactive command line (CLI)
 - openss -cli

```
Experiment Commands
expView
expCompare
expStatus
```

```
List Commands
list -v exp
list -v hosts
```

❖ Python module

```
import openss

my_filename=openss.FileList("myprog.a.out")
my_exptype=openss.ExpTypeList("pcsamp")
my_id=openss.expCreate(my_filename,my_exptype)

openss.expGo()

My_metric_list = openss.MetricList("exclusive")
my_viewtype = openss.ViewTypeList("pcsamp")
result = openss.expView(my_id,my_viewtype,my_metric_list)
```

❖ Users pick experiments:

- What to measure and from which sources?
- How to select, view, and analyze the resulting data?

❖ Two main classes:

- Statistical Sampling
 - Periodically interrupt execution and record location
 - Useful to get an overview
 - Low and uniform overhead
- Event Tracing
 - Gather and store individual application events
 - Provides detailed per event information
 - Can lead to huge data volumes

❖ O|SS can be extended with additional experiments

❖ PC Sampling (pcsamp)

- Record PC repeatedly at user defined time interval
- Low overhead overview of time distribution
- Good first step, lightweight overview

❖ Call Path Profiling (usertime)

- PC Sampling and Call stacks for each sample
- Provides inclusive and exclusive timing data
- Use to find hot call paths, whom is calling who

❖ Hardware Counters (hwc, hwctime, hwcsamp)

- Provides profile of hardware counter events like cache & TLB misses
- hwcsamp:
 - Periodically sample to capture profile of the code against the chosen counter
 - Default events are PAPI_TOT_INS and PAPI_TOT_CYC
- hwc, hwctime:
 - Sample a hardware counter till a certain number of events (called threshold) is recorded and get Call Stack
 - Default event is PAPI_TOT_CYC overflows

❖ **Input/Output Tracing (io, iot, iop)**

- Record invocation of all POSIX I/O events
- Provides aggregate and individual timings
- Store function arguments and return code for each call (iot)
- Lightweight I/O profiling because not tracking individual call details (iop)

❖ **MPI Tracing (mpi, mpit, mpip)**

- Record invocation of all MPI routines
- Provides aggregate and individual timings
- Store function arguments and return code for each call (mpit)
- Lightweight MPI profiling because not tracking individual call details (mpip)

❖ **Memory Tracing (mem)**

- Record invocation of key memory related function call events
- Provides aggregate and individual rank, thread, or process timings

❖ **CUDA NVIDIA GPU Event Tracing (cuda)**

- Record CUDA events, provides timeline and event timings
- Traces all NVIDIA CUDA kernel executions and the data transfers between main memory and the GPU.
- Records the call sites, time spent, and data transfer sizes.

❖ **POSIX thread tracing (pthreads)**

- Record invocation of all POSIX thread events
- Provides aggregate and individual rank, thread, or process timings

❖ **OpenMP specific profiling/tracing (omptp)**

- Report task idle, barrier, and barrier wait times per OpenMP thread and attribute those times to the OpenMP parallel regions.

1. Picking the experiment

- What do I want to measure?
- *We will start with pcsamp to get a first overview*

2. Launching the application

- How do I control my application under O|SS?
- *Enclose how you normally run your application in quotes*
- **osspcsamp** *"mpirun -np 4 smg2000 -n 65 65 65"*

3. Storing the results

- *O|SS will create a database*
- *Name: smg2000-pcsamp.openss*

4. Exploring the gathered data

- How do I interpret the data?
- *O|SS will print a default report (offline version only)*
- *Open the GUI to analyze data in detail (run: "**openss**")*

❖ osspcsamp “mpirun -np 4 smg2000 -n 65 65 65”

```
Bash> osspcsamp "mpirun -np 4 ./smg2000 -n 65 65 65"
```

```
[openss]: pcsamp experiment using the pcsamp experiment default sampling rate: "100".
```

```
[openss]: Using OPENSS_PREFIX installed in /opt/ossoffv2.1u4
```

```
[openss]: Setting up offline raw data directory in /opt/shared/offline-oss
```

```
[openss]: Running offline pcsamp experiment using the command:
```

```
"mpirun -np 4 /opt/ossoffv2.1u4/bin/ossrun " ./smg2000 -n 65 65 65" pcsamp"
```

Running with these driver parameters:

(nx, ny, nz) = (65, 65, 65)

...

<SMG native output>

...

Final Relative Residual Norm = 1.774415e-07

```
[openss]: Converting raw data from /opt/shared/offline-oss into temp file X.0.openss
```

Processing raw data for smg2000

Processing processes and threads ...

Processing performance data ...

Processing functions and statements ...

Resolving symbols for /home/jeg/DEMOS/workshop_demos/mpi/smg2000/test/smg2000

❖ osspcsamp “mpirun -np 4 smg2000 -n 65 65 65”

[openss]: Restoring and displaying default view for:

/home/jeg/DEMOS/workshop_demos/mpi/smg2000/test/smg2000-pcsamp.openss

[openss]: The restored experiment identifier is: -x 1

| Exclusive CPU time in seconds. | % of CPU Time | Function (defining location) |
|-----------------------------------|---------------|--|
| 7.870000 | 43.265531 | hypr_SMGResidual (smg2000: smg_residual.c,152) |
| 4.390000 | 24.134140 | hypr_CyclicReduction (smg2000: cyclic_reduction.c,757) |
| 1.090000 | 5.992303 | mca_btl_vader_check_fboxes (libmpi.so.1.4.0: btl_vader_fbox.h,108) |
| 0.510000 | 2.803738 | unpack_predefined_data (libopen-pal.so.6.1.1: opal_datatype_unpack.h,41) |
| 0.380000 | 2.089060 | hypr_SemiInterp (smg2000: semi_interp.c,126) |
| 0.360000 | 1.979109 | hypr_SemiRestrict (smg2000: semi_restrict.c,125) |
| 0.350000 | 1.924134 | __memcpy_ssse3_back (libc-2.17.so) |
| 0.310000 | 1.704233 | pack_predefined_data (libopen-pal.so.6.1.1: opal_datatype_pack.h,38) |
| 0.210000 | 1.154480 | hypr_SMGAxpy (smg2000: smg_axpy.c,27) |
| 0.140000 | 0.769654 | hypr_StructAxpy (smg2000: struct_axpy.c,25) |
| 0.110000 | 0.604728 | hypr_SMGSetStructVectorConstantValues (smg2000: smg.c,379) |
| | | |

❖ View with GUI: openss -f smg2000-pcsamp.openss

Default Output Report View



Toolbar to switch Views

Performance Data
Default view: by Function
(Data is sum from all processes and threads)
Select "Functions", click D-icon

Graphical Representation

pc Sampling [1]
Process Control
Run Cont Pause Update
Status: Process Loaded: Click on the "Run" button to begin the experiment.
Source Panel [1] Stats Panel [1]
View/Display Choice
Functions Statements Linked Objects Loops
Showing Functions Report:
Executables: smg2000 Host: localhost Pids: 4 Ranks: 4 Threads: 4

| % of CPU Time | Exclusive CPU time in seconds. | % of CPU Time | Function (defining location) |
|---------------|--------------------------------|---------------|--|
| 43.265531 | 7.870000 | 43.265531 | hypr_SMGResidual (smg2000: smg_residual.c,152) |
| 24.134140 | 4.390000 | 24.134140 | hypr_CyclicReduction (smg2000: cyclic_reduction.c,757) |
| 5.992303 | 1.090000 | 5.992303 | mca_btl_vader_check_fboxes (libmpi.so.1.4.0: btl_vader_fbox.h,108) |
| 2.803738 | 0.510000 | 2.803738 | unpack_predefined_data (libopen-pal.so.6.1.1: opal_datatype_unpack.h,41) |
| 2.089060 | 0.380000 | 2.089060 | hypr_SemiInterp (smg2000: semi_interp.c,126) |
| | 0.350000 | 1.979109 | hypr_SemiRestrict (smg2000: semi_restrict.c,125) |
| | 0.310000 | 1.924134 | __memcpy_ssse3_back (libc-2.17.so) |
| | 0.210000 | 1.704233 | pack_predefined_data (libopen-pal.so.6.1.1: opal_datatype_pack.h,38) |
| | | 1.154480 | hypr_SMGAXpr (smg2000: smg_axpr.c,27) |

Command Panel
opens>>

Statement Report Output View



Performance Data
View Choice: Statements
Select "statements, click D-icon

pc Sampling [1]

Process Control

Run Cont Pause Update Terminate

Status: Process Loaded: Click on the "Run" button to begin the experiment.

Source Panel [1] Stats Panel [1]

Showing Statements Report...

View Display Choice
☐ Function ☒ Statements ☐ Linked Objects ☐ Loops

Executables: smg2000 Host: localhost Pids: 4 Ranks: 4 Threads: 4

| % of CPU Time | Exclusive CPU time in seconds. | % of CPU Time | Statement Location (Line Number) |
|---------------|--------------------------------|---------------|----------------------------------|
| 38.438257 | 6.350000 | 38.438257 | smg_residual.c(291) |
| 8.353511 | 1.380000 | 8.353511 | cyclic_reduction.c(1130) |
| 5.569007 | 0.920000 | 5.569007 | smg_residual.c(239) |
| 5.387409 | 0.890000 | 5.387409 | cyclic_reduction.c(990) |
| 3.329298 | 0.550000 | 3.329298 | cyclic_reduction.c(999) |
| | 0.450000 | 2.723971 | bt_l_vader_fbox.h(121) |
| | 0.270000 | 1.634383 | cyclic_reduction.c(1061) |
| | 0.270000 | 1.634383 | semi_restrict.c(262) |
| other | 0.260000 | 1.573850 | cyclic_reduction.c(853) |

Command Panel

opens>>

Statement in Program that took the most time

Associate Source & Performance Data



Open|SpeedShop

File To

Process

Status: Process Loaded: Click on the "Run" button to begin the experiment.

Stats Panel [1]

View/Display

Function

Executables: smg2000 Host: localhost Pids: 4 Ranks: 4 Threads: 4

| Exclusive CPU time in seconds. | % of CPU Time | statement Location (Line Number) |
|--------------------------------|---------------|----------------------------------|
| 6.350000 | 38.438257 | smg_residual.c(291) |
| 1.380000 | 8.353511 | cyclic_reduction.c(1130) |
| 0.920000 | 5.569007 | smg_residual.c(239) |
| 0.890000 | 5.387409 | cyclic_reduction.c(910) |
| 0.550000 | 3.329298 | cyclic_reduction.c(999) |
| 0.450000 | 2.723971 | btl_vader_fbox.h(121) |
| 0.270000 | 1.634383 | cyclic_reduction.c(1061) |
| 0.270000 | 1.634383 | semi_restrict.c(262) |
| 0.260000 | 1.573850 | cyclic_reduction.c(853) |

Source Panel [1]

Exclusive CPU t

/home/jeg/DEMOS/test_workshop_demos/mpi/smg2000/struct_ls/smg_residual.c

```
283     A_data_box, start, base_stride, Ai,
284     x_data_box, start, base_stride, xi,
285     r_data_box, start, base_stride, ri);
286 #define HYPRE_BOX_SMP_PRIVATE loopk, loopi, loopj, Ai, xi, ri
287 #include "hypr_box_smp_forloop.h"
288
289     hypr_BoxLoop3For(loopi, loopj, loopk, Ai, xi, ri)
290     {
291         rp[ri] -= Ap[Ai] * xp[xi];
292     }
293     hypr_BoxLoop3End(Ai, xi, ri);
294 }
295 }
```

Command Panel

ManageProcessesPanel [1]

| Processes: | Rank | Thread | Status |
|------------|------|-----------------|--------------|
| 11531 | 0 | 139906999825280 | Disconnected |
| 11532 | 1 | 139901680468864 | Disconnected |
| 11533 | 2 | 140051826938752 | Disconnected |

Process Sets

PID

Rank

Thread

Dynamic I

All

Discon

Double click to open source window

Use window controls to split/arrange windows

Selected performance data point

Library (LinkedObject) View



Open|SpeedShop

File Tools Help

pc Sampling [1]

Process Control

Run Cont Pause Update

Status: Process Loaded: Click on the "Run" button to begin the experiment.

Source Panel [1] Stats Panel [1]

Showing Linked Objects Report...

View/Display Choice

☐ Functions ☐ Statements ☒ Linked Objects ☐ Loops

Executables: smg2000 Host: localhost Pids: 4 Ranks: 4 Threads: 4

| % of CPU Time | Exclusive CPU time in seconds. | % of CPU Time | LinkedObject |
|---------------|--------------------------------|---------------|----------------------|
| 78.143877 | 14.230000 | 78.143877 | smg2000 |
| 10.763317 | 1.960000 | 10.763317 | libmpi.so.1.4.0 |
| 7.138935 | 1.300000 | 7.138935 | libopen-pal.so.6.1.1 |
| 3.844042 | 0.700000 | 3.844042 | libc-2.17.so |
| | 0.010000 | 0.054915 | pcsamp-rt-offline.so |
| | 0.010000 | 0.054915 | libmonitor.so.0.0.0 |

Command Panel

opens>>

Select LinkedObject View type and Click on D-icon

Shows time spent in libraries. Can indicate imbalance.

Libraries in the application

Loop View



Open|SpeedShop

File Tools Help

pc Sampling [1]

Process Control

Run Cont Pause Update

Status: Process Loaded: Click on the "Run" button to begin the experiment.

Source Panel [1] Stats Panel [1]

Showing Loops Report...

View/Display Choice

☐ Functions ☐ Statements ☐ Linked Objects ☒ Loops

Executables: smg2000 Host: localhost Pids: 4 Ranks: 4 Threads: 4

| % of CPU Time | Exclusive CPU time in seconds. | % of CPU Time | Loop Start Location (Line Number) |
|---------------|--------------------------------|---------------|-----------------------------------|
| 28.754110 | 7.870000 | 28.754110 | smg_residual.c(205) |
| 7.672634 | 2.100000 | 7.672634 | cyclic_reduction.c(882) |
| 7.234198 | 1.980000 | 7.234198 | cyclic_reduction.c(1022) |
| 3.836317 | 1.050000 | 3.836317 | smg_residual.c(237) |
| 3.836317 | 1.050000 | 3.836317 | smg_residual.c(220) |
| 3.836317 | 1.050000 | 3.836317 | smg_residual.c(237) |
| 3.836317 | 1.040000 | 3.799781 | smg_residual.c(237) |
| other | 1.010000 | 3.690172 | btl_vader_fbox.h(117) |
| | 0.440000 | 1.607600 | opal_datatype_unpack.h(65) |

Command Panel

openss>>

Select Loops
View type and Click
on D-icon

Shows time spent in loops.

Statement number of start
of loop.

- ❖ **Place the way you run your application normally in quotes and pass it as an argument to osspcsamp, or any of the other experiment convenience scripts: ossio, ossmpi, etc.**
 - `osspcsamp "mpiexec_mpt -np 64 ./mpi_application app_args"`
- ❖ **Open | SpeedShop sends a summary profile to stdout**
- ❖ **Open | SpeedShop creates a database file**
- ❖ **Display alternative views of the data with the GUI via:**
 - `openss -f <database file>`
- ❖ **Display alternative views of the data with the CLI via:**
 - `openss -cli -f <database file>`
- ❖ **On clusters, need to set OPENSS_RAWDATA_DIR**
 - Only for the --offline mode of operation – not needed for default
 - Should point to a directory in a shared file system
 - Usually set/handled in a module or dotkit file.
- ❖ **Start with pcsamp for overview of performance**
- ❖ **Then, focus on performance issues with other experiments**

Flat Profile Overview

❖ Profiles show computationally intensive code regions

- First views: Time spent per functions or per statements

❖ Questions:

- Are those functions/statements expected?
- Do they match the computational kernels?
- Any runtime functions taking a lot of time?

❖ Identify bottleneck components

- View the profile aggregated by shared objects (LinkedObject view)
- Correct/expected modules?
- Impact of support and runtime libraries



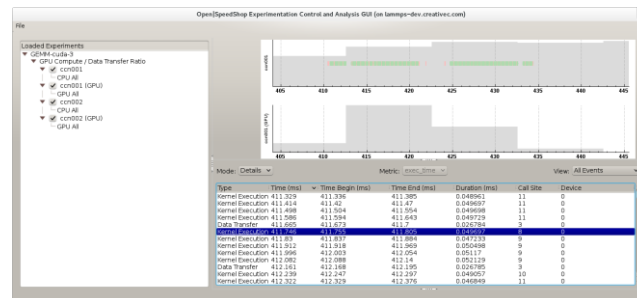
Open | SpeedShop™



Performance with Open/SpeedShop

NASA Open/SpeedShop Update/Training

Call Path Profiling (usertime)

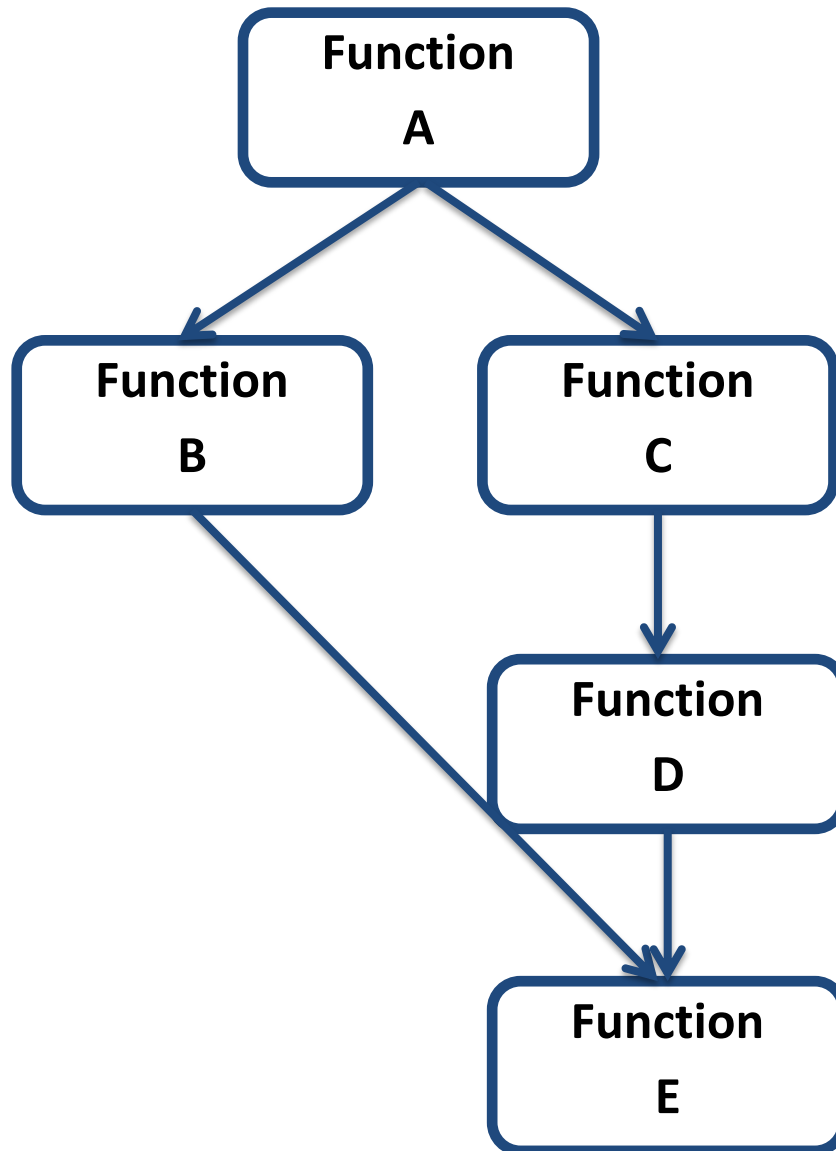


❖ Call Stack Profiling

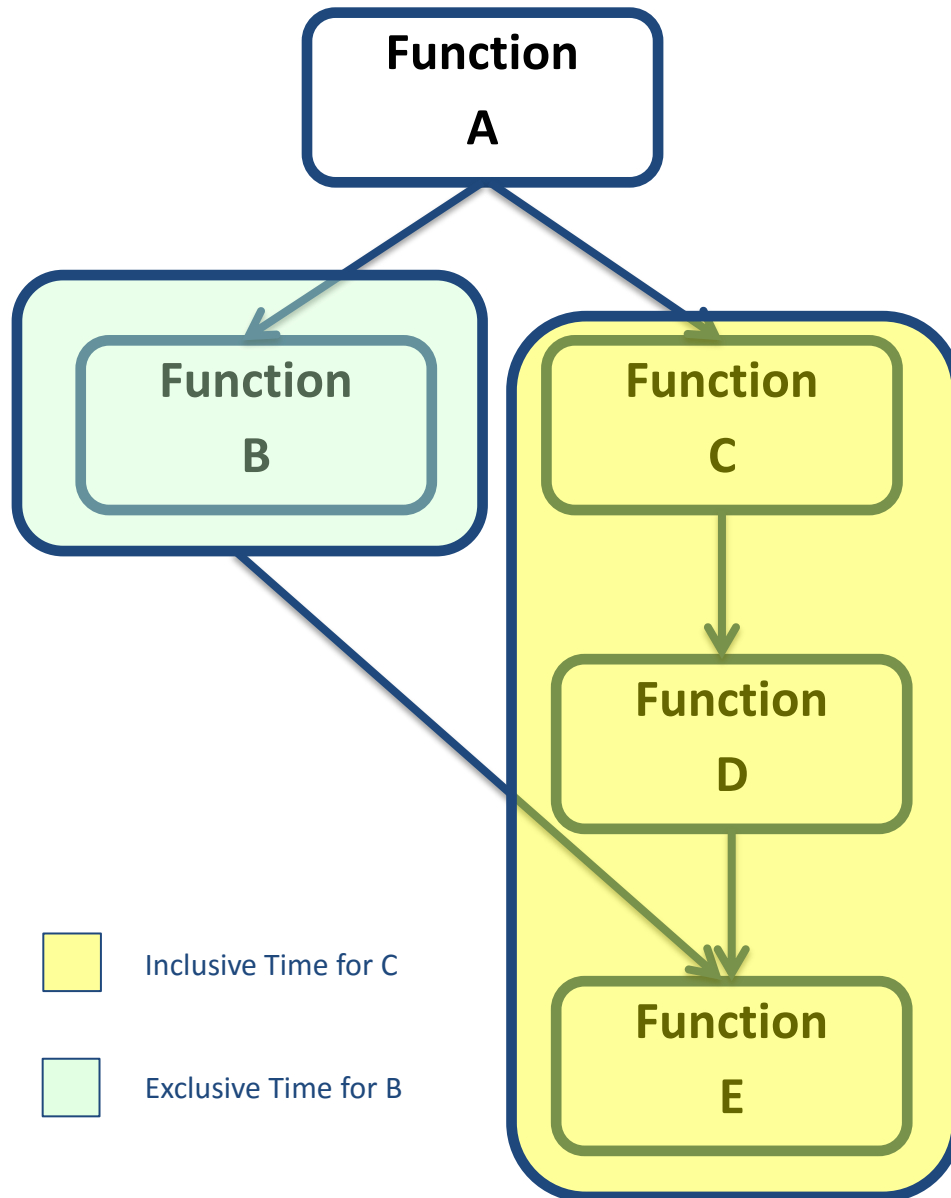
- Take a sample: address inside a function
- Call stack: series of program counter addresses (PCs)
- Unwinding the stack is walking through those addresses and recording that information for symbol resolution later.
- Leaf function is at the end of the call stack list

❖ Open | SpeedShop: experiment called usertime

- Time spent inside a routine vs. its children
- Time spent along call paths in the application
- Key view: butterfly



- ❖ **Missing information in flat profiles**
 - Distinguish routines called from multiple callers
 - Understand the call invocation history
 - Context for performance data
- ❖ **Critical technique: Stack traces**
 - Gather stack trace for each performance sample
 - Aggregate only samples with equal trace
- ❖ **User perspective:**
 - Butterfly views (caller/callee relationships)
 - Hot call paths
 - Paths through application that take most time



❖ Stack traces enable calculation of inclusive/exclusive times

- Time spent inside a function only (exclusive)
 - See: Function B
- Time spent inside a function and its children (inclusive)
 - See Function C and children

❖ Implementation similar to flat profiles

- Sample PC information
- Additionally collect call stack information at every sample

❖ Tradeoffs

- Pro: Obtain additional context information
- Con: Higher overhead/lower sampling rate

❖ Inclusive versus exclusive times

- If similar: child executions are insignificant
 - May not be useful to profile below this layer
- If inclusive time significantly greater than exclusive time:
 - Focus attention to the execution times of the children

❖ Hotpath analysis

- Which paths takes the most time?
- Path time might be ok & expected, but could point to a problem

❖ Butterfly analysis (similar to gprof)

- Could be done on “suspicious” functions
 - Functions with large execution time
 - Functions with large difference between implicit and explicit time
 - Functions of interest
 - Functions that “take unexpectedly long”
 - ...
- Shows split of time in callees and callers

Basic syntax:

ossustertime “how you run your executable normally”

Examples:

ossustertime “smg2000 -n 50 50 50”

ossustertime “smg2000 -n 50 50 50” low

❖ Parameters

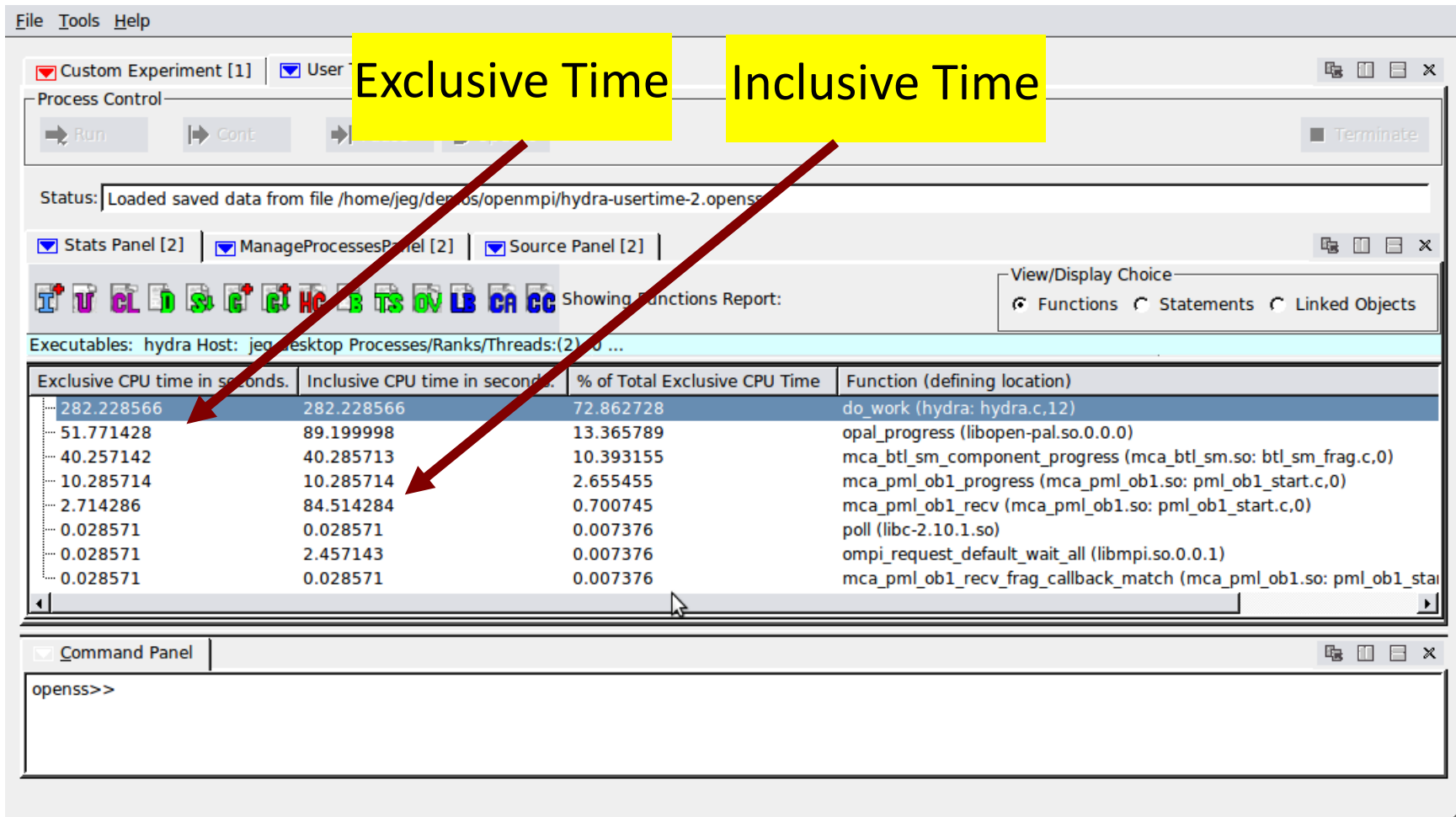
Sampling frequency (samples per second)

Alternative parameter: high (70) | low (18) | default (35)

Recommendation: compile code with -g to get statements!

❖ Default View

- Similar to pcsamp view from first example
- Calculates inclusive versus exclusive times



The screenshot displays the Open|SpeedShop interface. At the top, there are tabs for 'File', 'Tools', and 'Help'. Below this is a 'Process Control' section with buttons for 'Run', 'Cont', and 'Terminate'. The 'Status' bar indicates 'Loaded saved data from file /home/jeg/desktop/openmpi/hydra-usertime-2.openss'. The 'Stats Panel [2]' is active, showing a 'ManageProcessesPanel [2]' and a 'Source Panel [2]'. The 'View/Display Choice' dropdown is set to 'Functions'. The 'Executables' section shows 'hydra Host: jeg-desktop Processes/Ranks/Threads: (2) 0 ...'. The 'Showing Functions Report:' table is the central focus, displaying CPU times for various functions. Two yellow boxes with red arrows point to the 'Exclusive CPU time in seconds.' and 'Inclusive CPU time in seconds.' columns. The 'Command Panel' at the bottom shows 'openss>>'.

| Exclusive CPU time in seconds. | Inclusive CPU time in seconds. | % of Total Exclusive CPU Time | Function (defining location) |
|--------------------------------|--------------------------------|-------------------------------|--|
| 282.228566 | 282.228566 | 72.862728 | do_work (hydra: hydra.c,12) |
| 51.771428 | 89.199998 | 13.365789 | opal_progress (libopen-pal.so.0.0.0) |
| 40.257142 | 40.285713 | 10.393155 | mca_btl_sm_component_progress (mca_btl_sm.so: btl_sm_frag.c,0) |
| 10.285714 | 10.285714 | 2.655455 | mca_pml_ob1_progress (mca_pml_ob1.so: pml_ob1_start.c,0) |
| 2.714286 | 84.514284 | 0.700745 | mca_pml_ob1_recv (mca_pml_ob1.so: pml_ob1_start.c,0) |
| 0.028571 | 0.028571 | 0.007376 | poll (libc-2.10.1.so) |
| 0.028571 | 2.457143 | 0.007376 | ompi_request_default_wait_all (libmpi.so.0.0.1) |
| 0.028571 | 0.028571 | 0.007376 | mca_pml_ob1_recv_frag_callback_match (mca_pml_ob1.so: pml_ob1_stai |

Stack Trace Views: Hot Call Path



Open|SpeedShop

File Tools Help

☑ User Time [1]

Process Control

➡ Run ➡ Cont ➡ Pause ➡ Update

Status: Process Loaded: Click on the "Run" button to begin the experiment.

☑ Stats Panel [1] ☑ Manage Processes Panel [1]

Showing Hot Callpath Report:

Executables: smg2000 Host: localhost Pids: 2 Ranks: 2 Threads: 2

| Exclusive CPU time in seconds. | Inclusive CPU time in seconds. | % of Total Exclusive CPU | Call Stack Function (defining location) |
|--------------------------------|--------------------------------|--------------------------|---|
| | | | _start (smg2000) |
| | | | @ 556 in __libc_start_main (libmonitor.so.0.0.0) |
| | | | __libc_start_main (libc-2.14.90.so) |
| | | | @ 517 in monitor_main (libmonitor.so.0.0.0) |
| | | | @ 510 in main (smg2000: smg2000.c,21) |
| | | | @ 65 in HYPRE_StructSMGSolve (smg2000: HYPRE_struct_smg.c,64) |
| | | | @ 168 in hypre_SMGSolve (smg2000: smg_solve.c,57) |
| 0.171429 | 0.171429 | 1.754386 | @ 289 in hypre_SMGResidual (smg2000: smg_residual.c,152) |
| | | | _start (smg2000) |
| | | | @ 556 in __libc_start_main (libmonitor.so.0.0.0) |

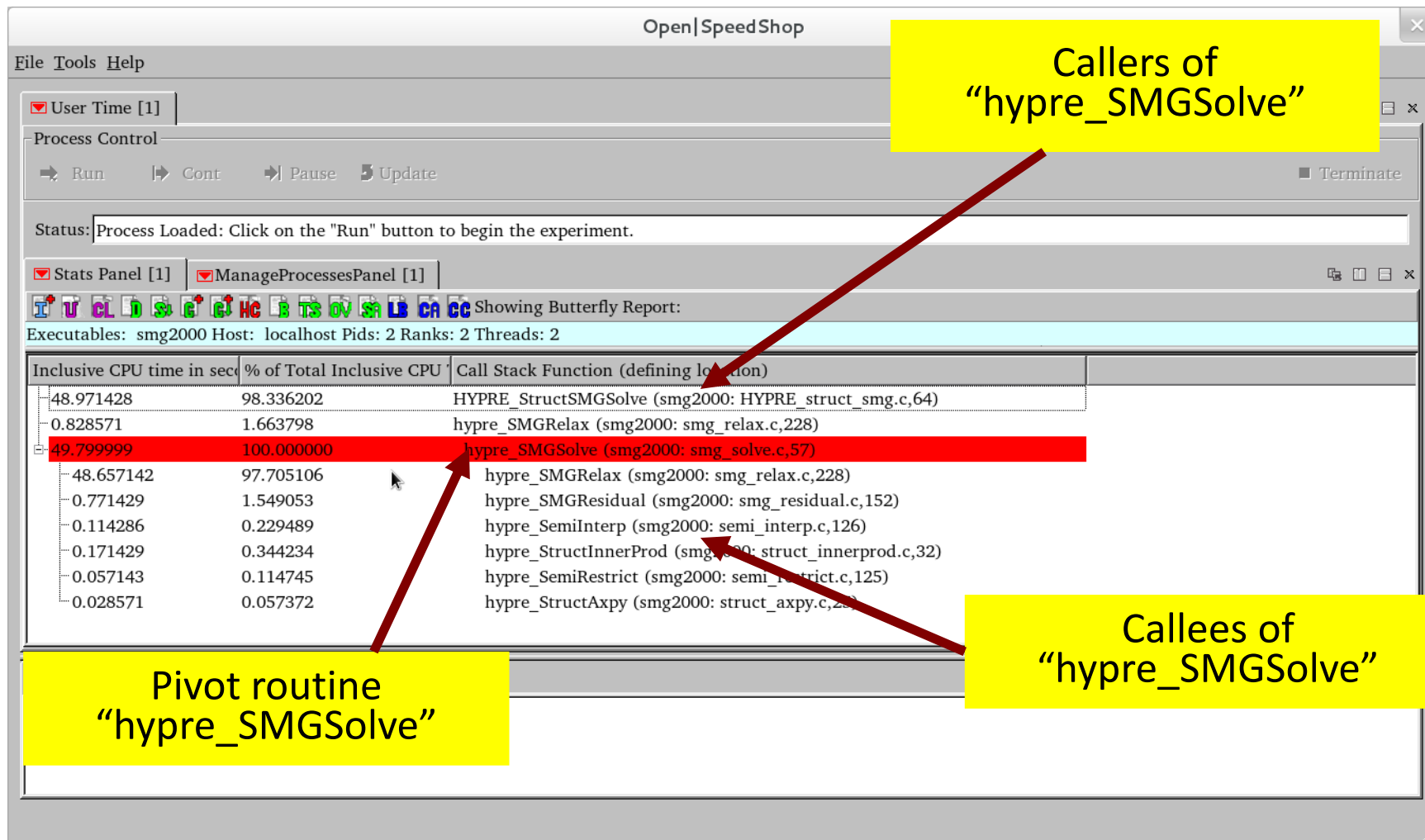
Access to call paths:

- All call paths (C+)
- All call paths for selected function (C↓)

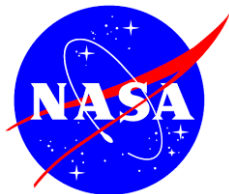
Stack Trace Views: Butterfly View



❖ Similar to well known “gprof” tool



- ❖ **Ustime experiment related application exercise**
- ❖ **Call path profiling exercises can be found in these directories:**
 - `$HOME/exercises/seq_smg2000`
 - `$HOME/exercises/smg2000`
 - `$HOME/exercises/lulesh2.0.3`



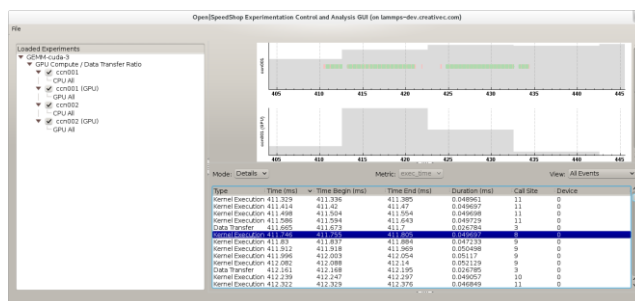
Open | SpeedShop™



Performance with Open/SpeedShop

NASA Open/SpeedShop Update/Training

Performance Analysis related to accessing
Hardware Counter Information



- ❖ **Timing information shows where you spend your time**
 - Hot functions / statements / libraries
 - Hot call paths
- ❖ **BUT: It doesn't show you why**
 - Are the computationally intensive parts efficient?
 - Are the processor architectural components working optimally?
- ❖ **Answer can be very platform dependent**
 - Bottlenecks may differ
 - Cause of missing performance portability
 - Need to tune to architectural parameters
- ❖ **Next: Investigate hardware/application interaction**
 - Efficient use of hardware resources or **Micro-architectural tuning**
 - Architectural units (on/off chip) that are stressed

❖ Provides access to hardware counters

- Implemented on top of PAPI
- Access to PAPI and native counters
- Examples: cache misses, TLB misses, bus accesses

❖ Basic model 1: Timer Based Sampling: hwcsamp

- Samples at set sampling rate for the chosen events
- Supports multiple counters
- Lower statistical accuracy
- Can be used to estimate good threshold for hwc/hwctime

❖ Basic model 2: Thresholding: hwc and hwctime

- User selects one counter
- Run until a fixed number of events have been reached
- Take PC sample at that location
 - **hwctime** also records stacktrace
- Reset number of events
- Ideal number of events (threshold) depends on application

- ❖ **osshwcsamp** “<command>< args>” [default |<PAPI_event_list>|<sampling_rate>]
 - Sequential job example:
osshwcsamp “smg2000”
 - Parallel job example:
osshwcsamp “mpirun -np 128 smg2000 -n 50 50 50” **PAPI_L1_DCM,PAPI_L1_TCA 50**
 - Default events: PAPI_TOT_CYC and PAPI_TOT_INS
 - Default sampling_rate: 100
 - <PAPI_event_list>: Comma separated PAPI event list (Maximum of 6 events that can be combined)
 - <sampling_rate>: Integer value sampling rate
- ❖ Use event count values to guide selection of thresholds for **hwc**, **hwctime** experiments for deeper analysis

Note: Counts in hwcsamp are the counts for all code executed in the sample period and therefore some counters may not have affected the count at the sampled point. **(fp ops counts at some obvious function or statement that does not have fp instructions)**

❖ For osshwcsamp, Open | SpeedShop supports ...

- Derived and Non derived PAPI presets
 - All derived and non derived events reported by “papi_avail”
 - Ability to sample up to six (6) counters at one time; before use test with
 - **papi_event_choser PRESET <list of events>**
 - If a counter does not appear in the output, there may be a conflict in the hardware counters
- All native events
 - Architecture specific (incl. naming)
 - Names listed in the PAPI documentation
 - Native events reported by “papi_native_avail”

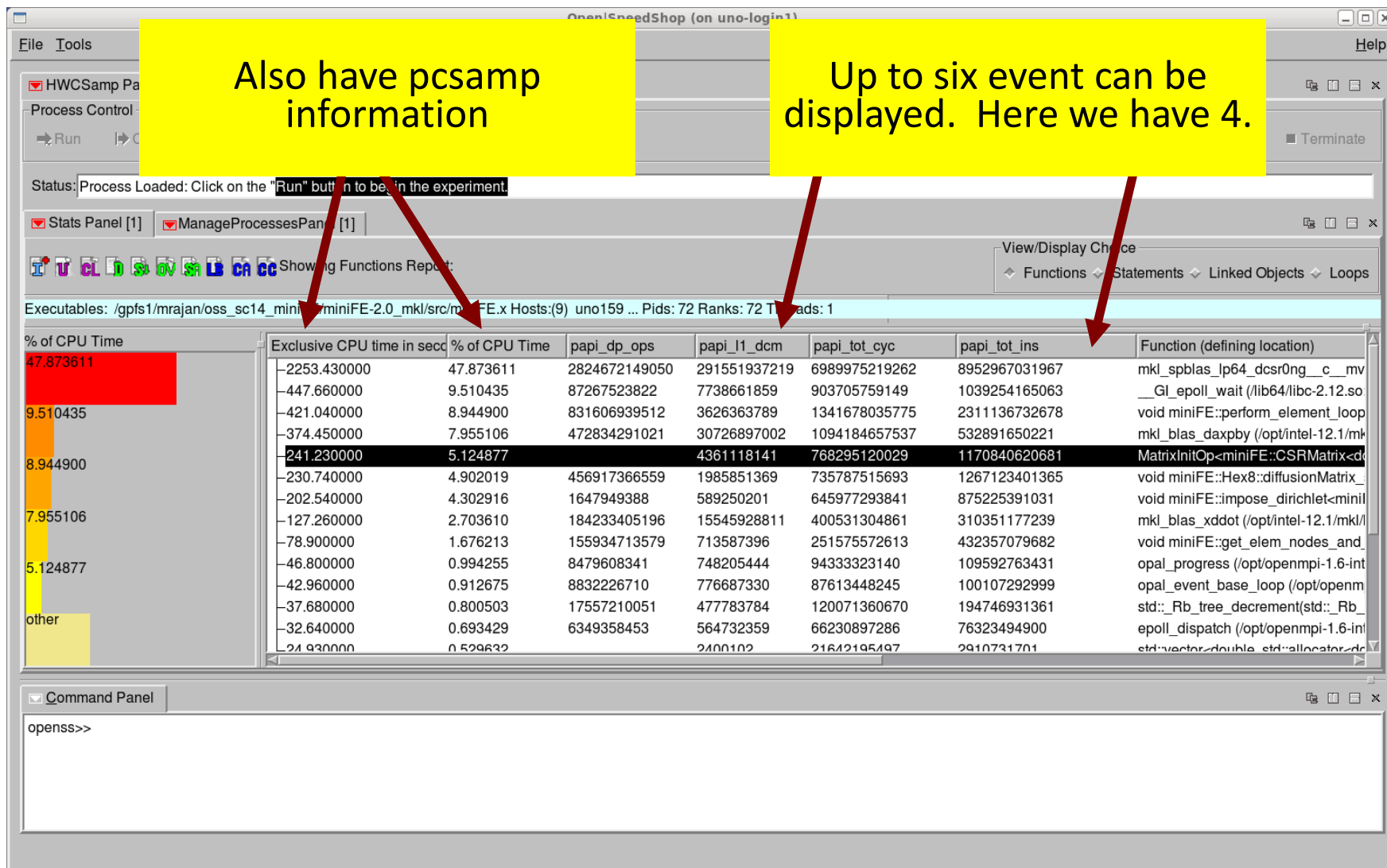
❖ Sampling rate depends on application

- Overhead vs. Accuracy
 - Lower sampling rate cause less samples

hwcsamp with miniFE (see mantevo.org)



- ❖ `osshwcsamp "mpiexec -n 72 miniFE.X -nx 614 -ny 614 -nz 614" PAPI_DP_OPS,PAPI_L1_DCM,PAPI_TOT_CYC,PAPI_TOT_INS`
- ❖ `openss -f miniFE.x-hwcsamp.openss`



- ❖ **osshwc[time]** “<command> < args>” [default | <PAPI_event> | <PAPI threshold> | <PAPI_event><PAPI threshold>]
 - Sequential job example:
 - **osshwc[time]** “smg2000 -n 50 50 50” PAPI_FP_OPS 50000
 - Parallel job example:
 - **osshwc[time]** “mpirun -np 128 smg2000 -n 50 50 50”
- ❖ **default:** event (PAPI_TOT_CYC), threshold (10000)
- ❖ <PAPI_event>: PAPI event name
- ❖ <PAPI threshold>: PAPI integer threshold
- ❖ **NOTE:** If the output is empty, try lowering the <threshold> value. There may not have been enough PAPI or native event occurrences to record and present

Examples of Typical Counters



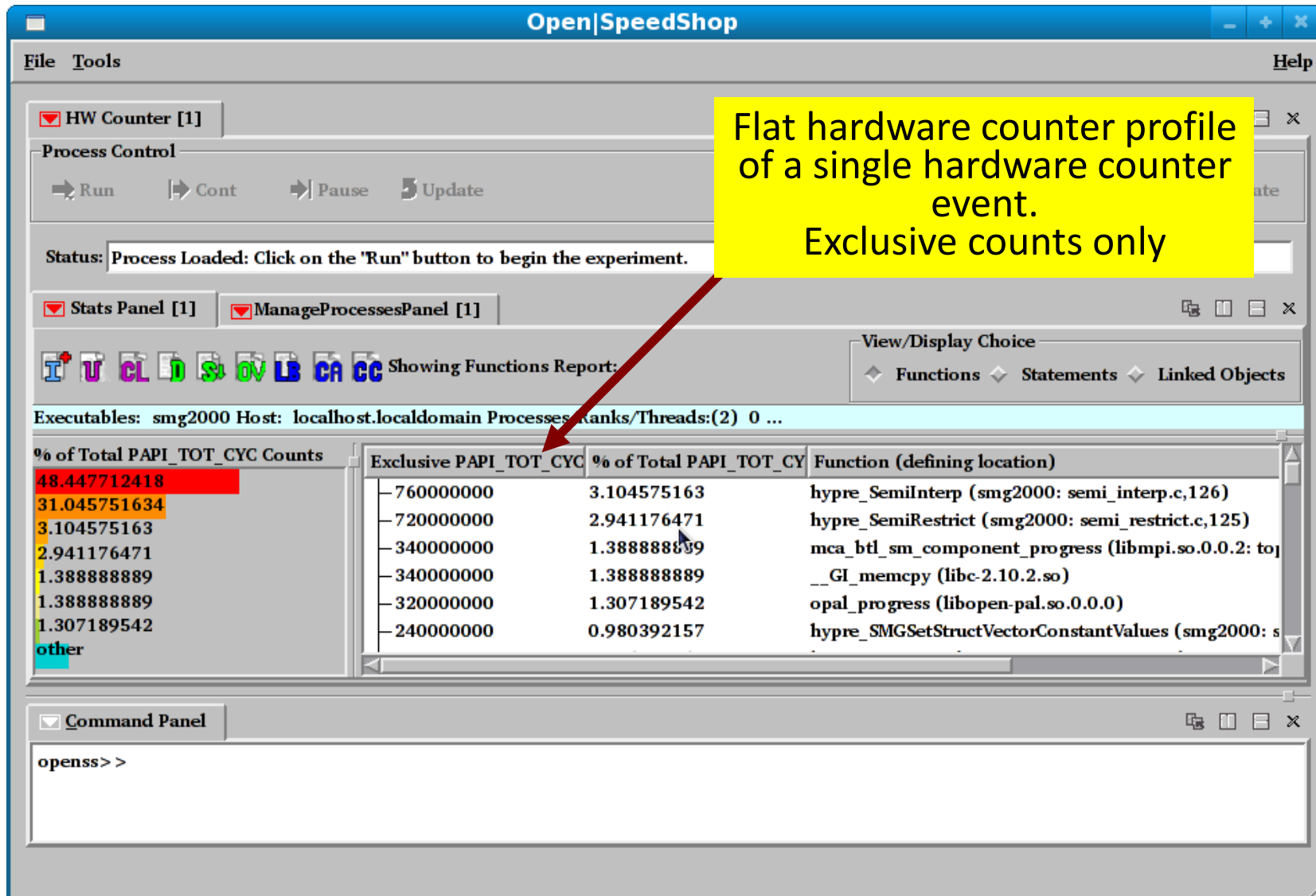
| PAPI Name | Description | Threshold |
|--------------|---------------------------------------|-------------|
| PAPI_L1_DCM | L1 data cache misses | high |
| PAPI_L2_DCM | L2 data cache misses | high/medium |
| PAPI_L1_DCA | L1 data cache accesses | high |
| PAPI_FPU_IDL | Cycles in which FPUs are idle | high/medium |
| PAPI_STL_ICY | Cycles with no instruction issue | high/medium |
| PAPI_BR_MSP | Miss-predicted branches | medium/low |
| PAPI_FP_INS | Number of floating point instructions | high |
| PAPI_LD_INS | Number of load instructions | high |
| PAPI_VEC_INS | Number of vector/SIMD instructions | high/medium |
| PAPI_HW_INT | Number of hardware interrupts | low |
| PAPI_TLB_TL | Number of TLB misses | low |

Note: Threshold indications are just rough guidance and depend on the application.

Note: Counters platform dependent (use **papi_avail** & **papi_native_avail**)

Note: The best way to choose a threshold for events is to observe the summary in hwcsamp for a counter and choose a value based on some small percentage of that.

❖ hwc default view: Counter = Total Cycles



The screenshot shows the Open|SpeedShop application window. The 'HW Counter [1]' panel is active, displaying a 'Process Control' section with 'Run', 'Cont', 'Pause', and 'Update' buttons. Below this is a 'Status' bar indicating 'Process Loaded: Click on the "Run" button to begin the experiment.' The 'Stats Panel [1]' and 'ManageProcessesPanel [1]' are also visible. The 'Showing Functions Report:' section displays a table of hardware counter data. A yellow callout box points to the 'Exclusive PAPI_TOT_CYC' column, stating: 'Flat hardware counter profile of a single hardware counter event. Exclusive counts only'.

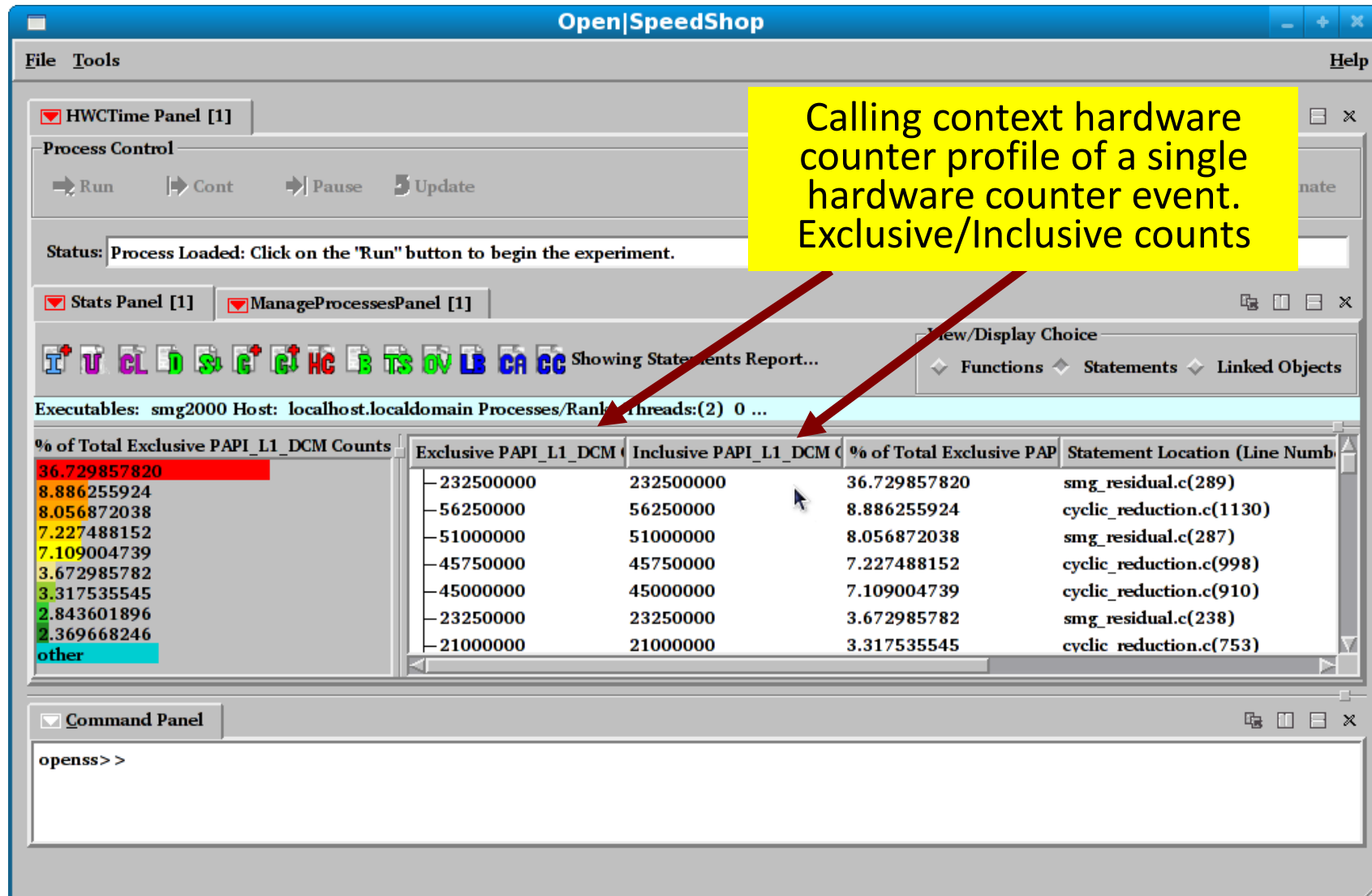
| % of Total PAPI_TOT_CYC Counts | Exclusive PAPI_TOT_CYC | % of Total PAPI_TOT_CYC | Function (defining location) |
|--------------------------------|------------------------|-------------------------|---|
| 48.447712418 | 760000000 | 3.104575163 | hypr_SemiInterp (smg2000: semi_interp.c,126) |
| 31.045751634 | 720000000 | 2.941176471 | hypr_SemiRestrict (smg2000: semi_restrict.c,125) |
| 3.104575163 | 340000000 | 1.388888889 | mca_btl_sm_component_progress (libmpi.so.0.0.2: to) |
| 2.941176471 | 340000000 | 1.388888889 | __GI_memcpy (libc-2.10.2.so) |
| 1.388888889 | 320000000 | 1.307189542 | opal_progress (libopen-pal.so.0.0.0) |
| 1.388888889 | 240000000 | 0.980392157 | hypr_SMGSetStructVectorConstantValues (smg2000: s |
| 1.307189542 | | | |
| other | | | |

Executables: smg2000 Host: localhost.localdomain Processes/Ranks/Threads:(2) 0 ...

View/Display Choice: Functions Statements Linked Objects

Command Panel: openss> >

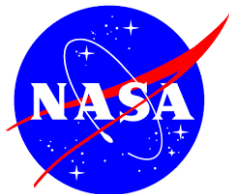
hwctime default view: Counter = L1 Cache Misses



The screenshot shows the Open|SpeedShop application window. The 'HWTime Panel [1]' is active, displaying a 'Process Control' section with buttons for Run, Cont, Pause, and Update. Below this is a 'Status' bar indicating 'Process Loaded: Click on the "Run" button to begin the experiment.' The 'Stats Panel [1]' and 'ManageProcessesPanel [1]' are also visible. The 'Showing Statements Report...' section displays a table of hardware counter data. A yellow callout box with red arrows points to the 'Exclusive PAPI_L1_DCM' and 'Inclusive PAPI_L1_DCM' columns, stating: 'Calling context hardware counter profile of a single hardware counter event. Exclusive/Inclusive counts'.

| % of Total Exclusive PAPI_L1_DCM Counts | Exclusive PAPI_L1_DCM | Inclusive PAPI_L1_DCM | % of Total Exclusive PAPI_L1_DCM | Statement Location (Line Number) |
|---|-----------------------|-----------------------|----------------------------------|----------------------------------|
| 36.729857820 | 232500000 | 232500000 | 36.729857820 | smg_residual.c(289) |
| 8.886255924 | 56250000 | 56250000 | 8.886255924 | cyclic_reduction.c(1130) |
| 8.056872038 | 51000000 | 51000000 | 8.056872038 | smg_residual.c(287) |
| 7.227488152 | 45750000 | 45750000 | 7.227488152 | cyclic_reduction.c(998) |
| 7.109004739 | 45000000 | 45000000 | 7.109004739 | cyclic_reduction.c(910) |
| 3.672985782 | 23250000 | 23250000 | 3.672985782 | smg_residual.c(238) |
| 3.317535545 | 21000000 | 21000000 | 3.317535545 | cyclic_reduction.c(753) |
| other | | | | |

Command Panel: opens>>



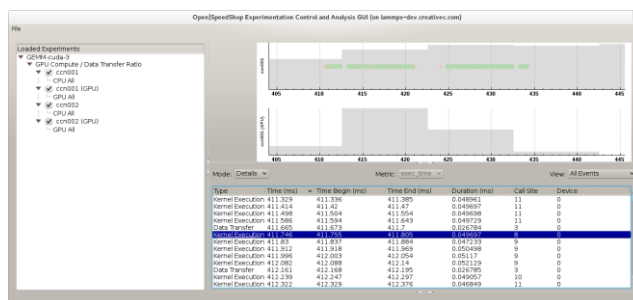
Open | SpeedShop™

Performance with Open/SpeedShop



NASA Open/SpeedShop Update/Training

Performance Analysis related to
application I/O activity



❖ I/O Tracing (io experiment)

- Records each event in chronological order
- Provides call path and time spent in I/O functions

❖ I/O Profiling (iop experiment)

- Lighter weight I/O tracking experiment
- Trace I/O functions but only record individual callpaths not each individual event with callpath (Like usertime)

❖ Extended I/O Tracing (iot experiment)

- Records each event in chronological order
- Collects Additional Information
 - Function Parameters
 - Function Return Value
- When to use extended I/O tracing?
 - When you want to trace the exact order of events
 - When you want to see the return values or bytes read or written.
 - When you want to see the parameters of the IO call

io/iop/iot experiment syntax and examples

Convenience script basic syntax:

ossio[p][t] “executable” [default | <list of I/O func>]

➤ Parameters

- I/O Function list to sample(default is all)
- creat, creat64, dup, dup2, lseek, lseek64, open, open64, pipe, pread, pread64, pwrite, pwrite64, read, readv, write, writev

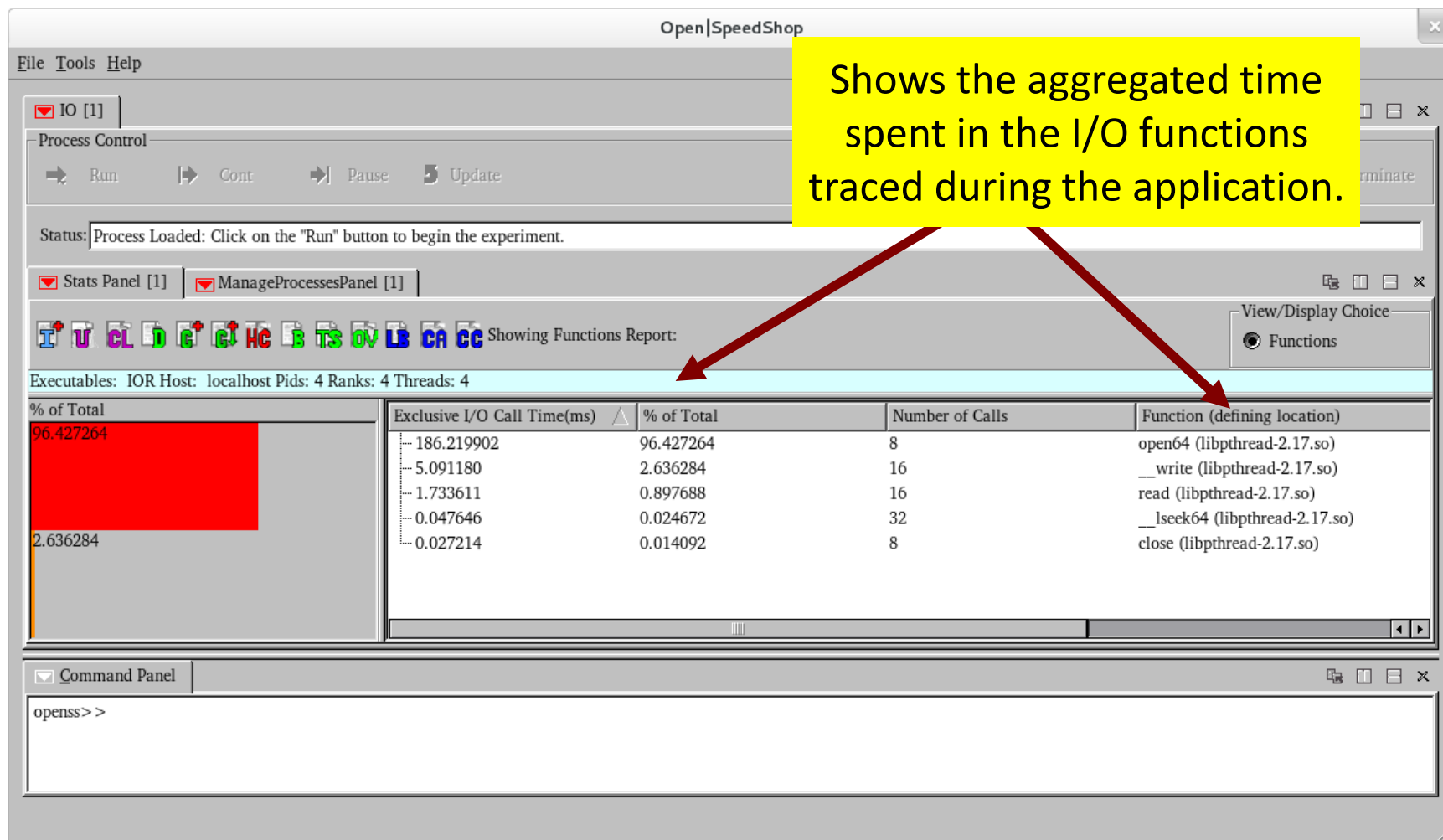
Examples:

ossio “mpirun -np 256 sweep3d.mpi”

ossiop “mpirun -np 256 sweep3d.mpi” read,readv,write

ossiott “mpirun -np 256 sweep3d.mpi” read,readv,write

❖ I/O Default View for IOR application “io” experiment

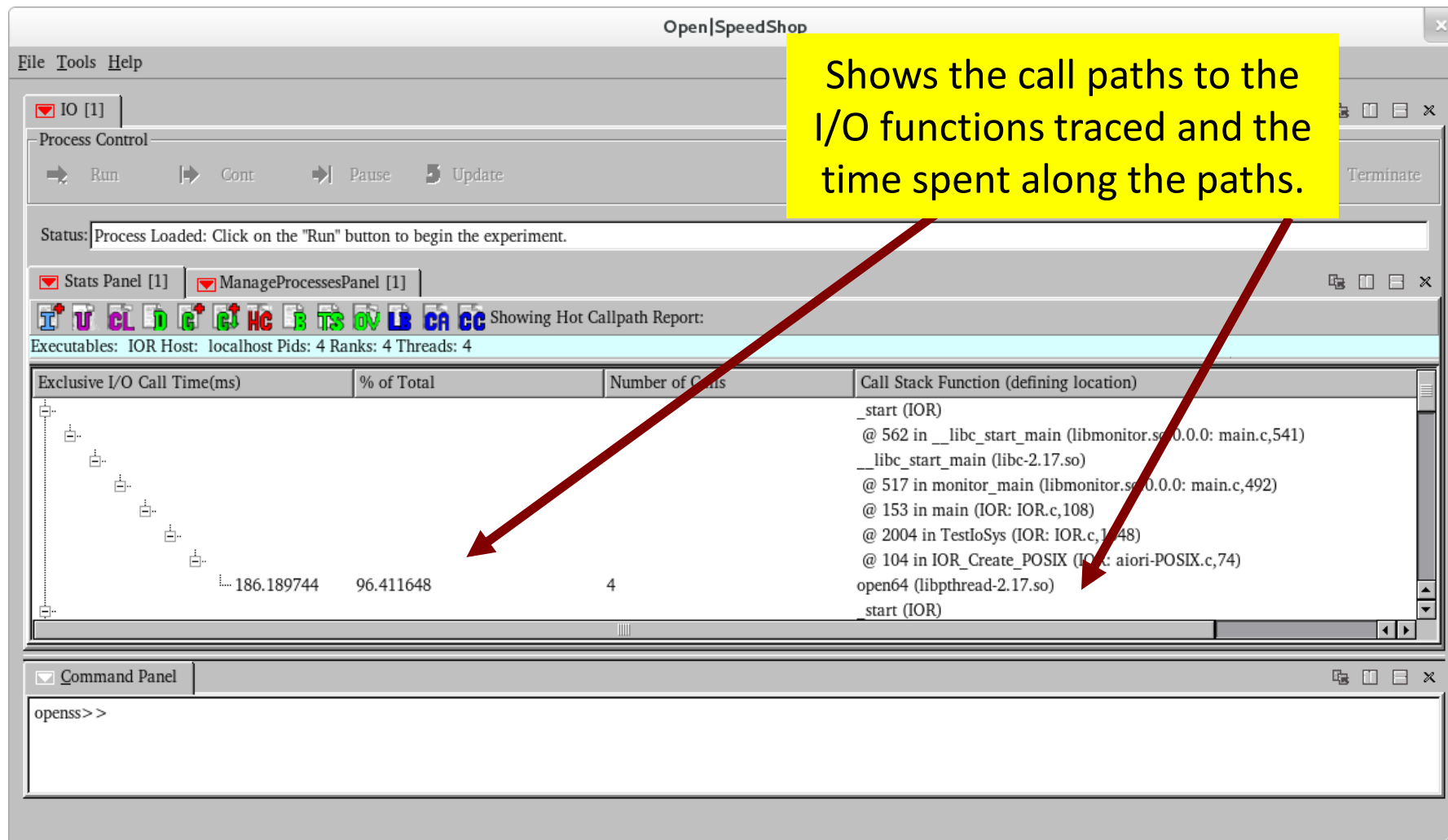


Shows the aggregated time spent in the I/O functions traced during the application.

Executables: IOR Host: localhost Pids: 4 Ranks: 4 Threads: 4

| Exclusive I/O Call Time(ms) | % of Total | Number of Calls | Function (defining location) |
|-----------------------------|------------|-----------------|--------------------------------|
| 186.219902 | 96.427264 | 8 | open64 (libpthread-2.17.so) |
| 5.091180 | 2.636284 | 16 | __write (libpthread-2.17.so) |
| 1.733611 | 0.897688 | 16 | read (libpthread-2.17.so) |
| 0.047646 | 0.024672 | 32 | __lseek64 (libpthread-2.17.so) |
| 0.027214 | 0.014092 | 8 | close (libpthread-2.17.so) |

❖ I/O Call Path View for IOR application “io” experiment



Shows the call paths to the I/O functions traced and the time spent along the paths.

| Exclusive I/O Call Time(ms) | % of Total | Number of Calls | Call Stack Function (defining location) |
|-----------------------------|------------|-----------------|--|
| 186.189744 | 96.411648 | 4 | <ul style="list-style-type: none">_start (IOR)@ 562 in __libc_start_main (libmonitor.so.0.0.0: main.c,541)__libc_start_main (libc-2.17.so)@ 517 in monitor_main (libmonitor.so.0.0.0: main.c,492)@ 153 in main (IOR: IOR.c,108)@ 2004 in TestIoSys (IOR: IOR.c,1048)@ 104 in IOR_Create_POSIX (IOR: aiori-POSIX.c,74)open64 (libpthread-2.17.so)_start (IOR) |

❖ I/O Default View for IOR application “iot” experiment

Open|SpeedShop (on rzmerl156)

File Tools

IOT [1]

Process Control

Run Cont Pause Update

Status: Process Loaded: Click on the "Run" button to begin the experiment.

Stats Panel [1] ManageProcessesPanel [1]

Showing Functions Report:

Executables: IOR Hosts:(32) rzmerl100 ... Pids: 512 Ranks: 512 Threads: 512

View/Display Choice
Functions

| I/O Call Time(ms) | % of Total I/O Time | Number of Calls | Min_Bytes Count | Min_Bytes Read Written | Max_Bytes Count | Max_Bytes Read Written | Total_Bytes Read Written | Function (defining location) |
|-------------------|---------------------|-----------------|-----------------|------------------------|-----------------|------------------------|--------------------------|--|
| 1858436.714506 | 61.486889 | 2048 | | | | | | close (libc-2.12.so: syscall-template.S,82) |
| 1055603.730633 | 34.924939 | 2048 | 2048 | 262144 | 2048 | 262144 | 536870912 | __GI__read (libc-2.12.so: syscall-template.S,82) |
| 108107.666680 | 3.576772 | 1024 | | | | | | __libc_open (libc-2.12.so: syscall-template.S,82) |
| 335.820251 | 0.011111 | 3072 | 512 | 6 | 2048 | 262144 | 536878080 | write (libc-2.12.so: syscall-template.S,82) |
| 8.756634 | 0.000290 | 4096 | | | | | | __GI__libc_lseek (libc-2.12.so: syscall-template.S,82) |

Command Panel

openss>>

Shows the min and max values for bytes read or written.

- ❖ Show the call paths in the application run that allocated the largest number of bytes
- ❖ Using the min_bytes would show all the paths that allocated the minimum number of bytes.
- ❖ openss>>expview -vcalltrees,fullstack -m max_bytes
- ❖ Max_Bytes Call Stack Function (defining location)
- ❖ Read
- ❖ Written
 - _start (IOR)
 - > @ 562 in __libc_start_main (libmonitor.so.0.0.0: main.c,541)
 - >> @ 258 in __libc_start_main (libc-2.12.so: libc-start.c,96)
 - >>> @ 517 in monitor_main (libmonitor.so.0.0.0: main.c,492)
 - >>>> @ 153 in main (IOR: IOR.c,108)
 - >>>>> @ 2013 in TestloSys (IOR: IOR.c,1848)
 - >>>>>> @ 2608 in WriteOrRead (IOR: IOR.c,2562)
 - >>>>>>> @ 244 in IOR_Xfer_POSIX (IOR: aiori-POSIX.c,224)
 - >>>>>>>> @ 321 in write (iot-collector-monitor-mrnet-mpi.so: wrappers.c,239)
- ❖ 262144 >>>>>>>>> @ 82 in write (libc-2.12.so: syscall-template.S,82)



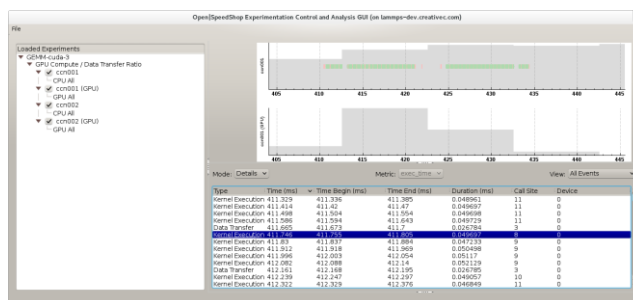
Open | SpeedShop™

Performance with Open/SpeedShop



NASA Open/SpeedShop Update/Training

Parallel Performance Analysis including
analysis related to application
MPI and/or OpenMP activity



❖ **O|SS is designed to work on parallel jobs**

- Support for threading and message passing
- Automatically tracks all ranks and threads during execution
- Records/stores performance info per process/rank/thread

❖ **All experiments can be used on parallel jobs**

- O|SS applies the experiment collector to all ranks or threads on all nodes

❖ **MPI specific tracing experiments**

- Tracing of MPI function calls (individual, all, or a specific group)
- Four forms of MPI tracing experiments

❖ **OpenMP specific experiment (ossomptp)**

- Uses OMPT API to record task time, idleness, barrier, and wait barrier per OpenMP parallel region
 - Shows load balance for time
 - expcompare time across all threads

❖ Viewing data from parallel codes

- By default all values aggregated (summed) across all ranks
- Manually include/exclude individual ranks/processes/threads
- Ability to compare ranks/threads

❖ Additional analysis options

- Load Balance (min, max, average) across parallel executions
 - Across ranks for hybrid OpenMP/MPI codes
 - Focus on a single rank to see load balance across OpenMP threads
- Cluster analysis (finding outliers)
 - Automatically creates groups of similar performing ranks or threads
 - Available from the Stats Panel toolbar or context menu
 - Note: can take a long time for large numbers of processors (current version)

❖ **O|SS has been tested with a variety of MPIs**

- Including: Open MPI, MVAPICH[2], and MPICH (Intel, Cray), MPT (SGI)

❖ **Running O|SS experiments on MPI codes**

- Just use the convenience script corresponding to the data you want to gather and put the command you use to run your application in quotes:
 - **osspsamp** “mpirun -np 32 sweep3d.mpi”
 - **ossio** “srun -N 4 -n 16 sweep3d.mpi”
 - **osshwctime** “mpirun -np 128 sweep3d.mpi”
 - **ossusertime** “srun -N 8 -n 128 sweep3d.mpi”
 - **osshwc** “mpirun -np 128 sweep3d.mpi”

❖ MPI specific experiments

- Record MPI call invocations – 100 or so that O|SS traces
- MPI functions are profiled (**ossmpip**)
 - Show call paths for each MPI unique call path
 - However individual call information is not recorded.
 - Less overhead than mpi, mpit.
- MPI functions are traced (**ossmpi**)
 - Record call times and call paths for each event
- MPI functions are traced with details (**ossmpit**)
 - Record call times, call paths and argument info for each event

❖ OpenMP specific experiment (**ossomptp**)

- Uses OMPT API to record task time, idleness, barrier, and wait barrier per OpenMP parallel region
 - Shows load balance for time
 - Can use CLI command: **expcompare** to compare time across all threads

mpi/mpip/mpit experiment syntax and examples

Convenience script basic syntax:

ossmpi[t][p] “mpi executable syntax” [default | <list MPI func> | mpi category]

➤ Parameters

- Default is all MPI Functions Open | SpeedShop traces
- MPI Function list to trace (comma separated)
 - MPI_Send, MPI_Recv,
- mpi_category:
 - "all", "asynchronous_p2p", "collective_com", "datatypes", "environment", "graphs_contexts_comms", "persistent_com", "process_topologies", "synchronous_p2p"

Examples:

ossmpi “srun -N 4 -n 32 smg2000 -n 50 50 50”

ossmpi “mpirun -np 4000 nbody” MPI_Send, MPI_Recv

❖ Get overview of application

- Run a lightweight experiment to verify performance expectations
 - pcsamp, usertime, hwc

❖ Use load balance view on pcsamp, usertime, hwc

- Look for performance values outside of norm
 - Somewhat large difference for the min, max, average values
 - If the MPI libraries are showing up in the load balance for pcsamp, then do an MPI specific experiment

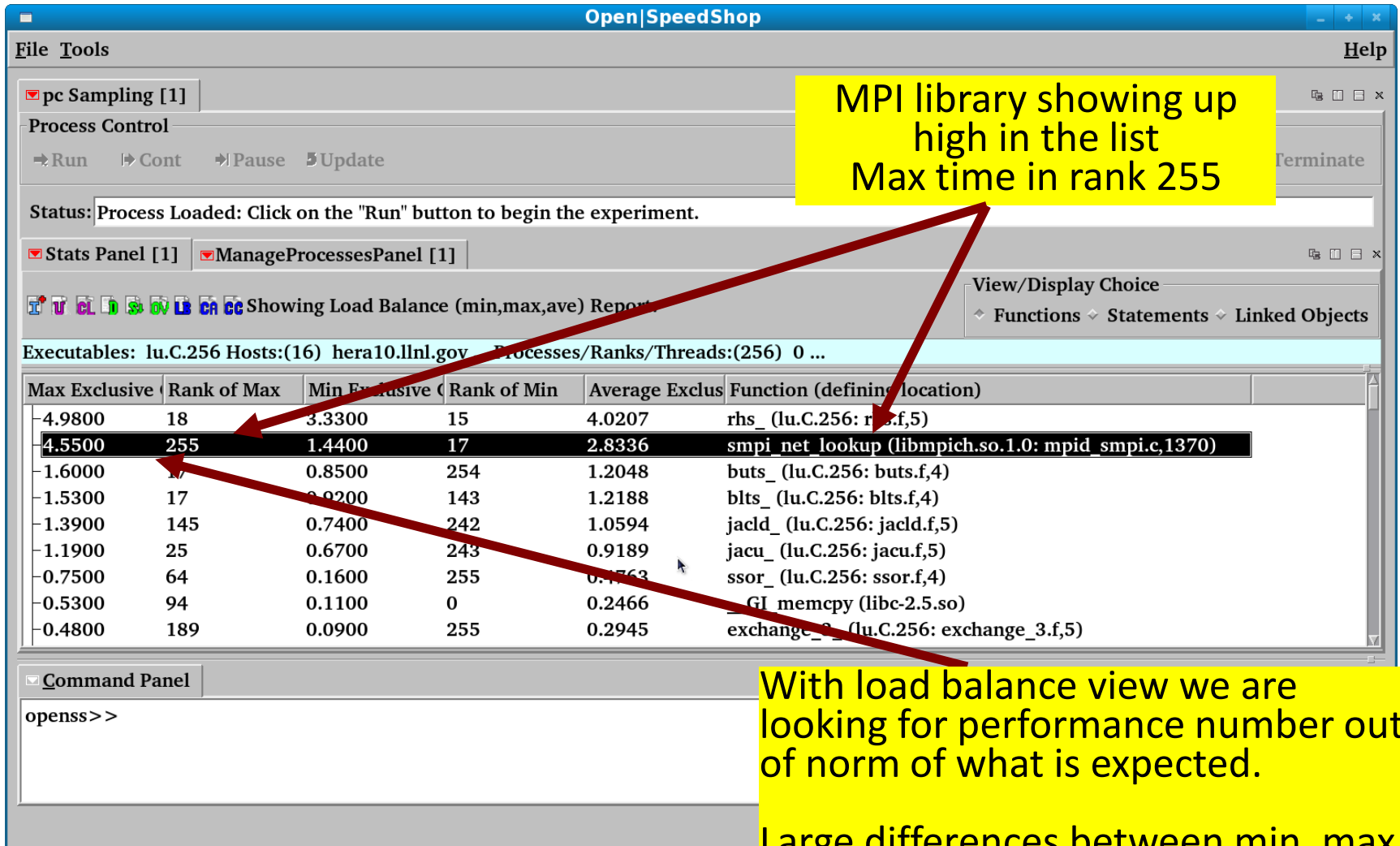
❖ Use load balance view on MPI experiment

- Look for performance values outside of norm
 - Somewhat large difference for the min, max, average values
- Focus on the MPI functions to find potential problems

❖ Use load balance view on OpenMP experiment (omptp)

- Can also use expcompare across OpenMP threads

❖ Load Balance View based on functions (pcsamp)



Open|SpeedShop

File Tools Help

☒ pc Sampling [1]

Process Control

→ Run → Cont → Pause → Update

Status: Process Loaded: Click on the "Run" button to begin the experiment.

☒ Stats Panel [1] ☒ ManageProcessesPanel [1]

Showing Load Balance (min,max,ave) Report

Executables: lu.C.256 Hosts:(16) hera10.llnl.gov Processes/Ranks/Threads:(256) 0 ...

| Max Exclusive | Rank of Max | Min Exclusive | Rank of Min | Average Excl | Function (defining location) |
|---------------|-------------|---------------|-------------|--------------|---|
| 4.9800 | 18 | 3.3300 | 15 | 4.0207 | rhs_ (lu.C.256: rhs.f,5) |
| 4.5500 | 255 | 1.4400 | 17 | 2.8336 | smpi_net_lookup (libmpich.so.1.0: mpid_smpi.c,1370) |
| 1.6000 | 17 | 0.8500 | 254 | 1.2048 | buts_ (lu.C.256: buts.f,4) |
| 1.5300 | 17 | 0.9200 | 143 | 1.2188 | blts_ (lu.C.256: blts.f,4) |
| 1.3900 | 145 | 0.7400 | 242 | 1.0594 | jacld_ (lu.C.256: jacld.f,5) |
| 1.1900 | 25 | 0.6700 | 243 | 0.9189 | jacu_ (lu.C.256: jacu.f,5) |
| 0.7500 | 64 | 0.1600 | 255 | 0.4763 | ssor_ (lu.C.256: ssor.f,4) |
| 0.5300 | 94 | 0.1100 | 0 | 0.2466 | _GI_memcpy (libc-2.5.so) |
| 0.4800 | 189 | 0.0900 | 255 | 0.2945 | exchange_3_ (lu.C.256: exchange_3.f,5) |

View/Display Choice

◆ Functions ◆ Statements ◆ Linked Objects

☒ Command Panel

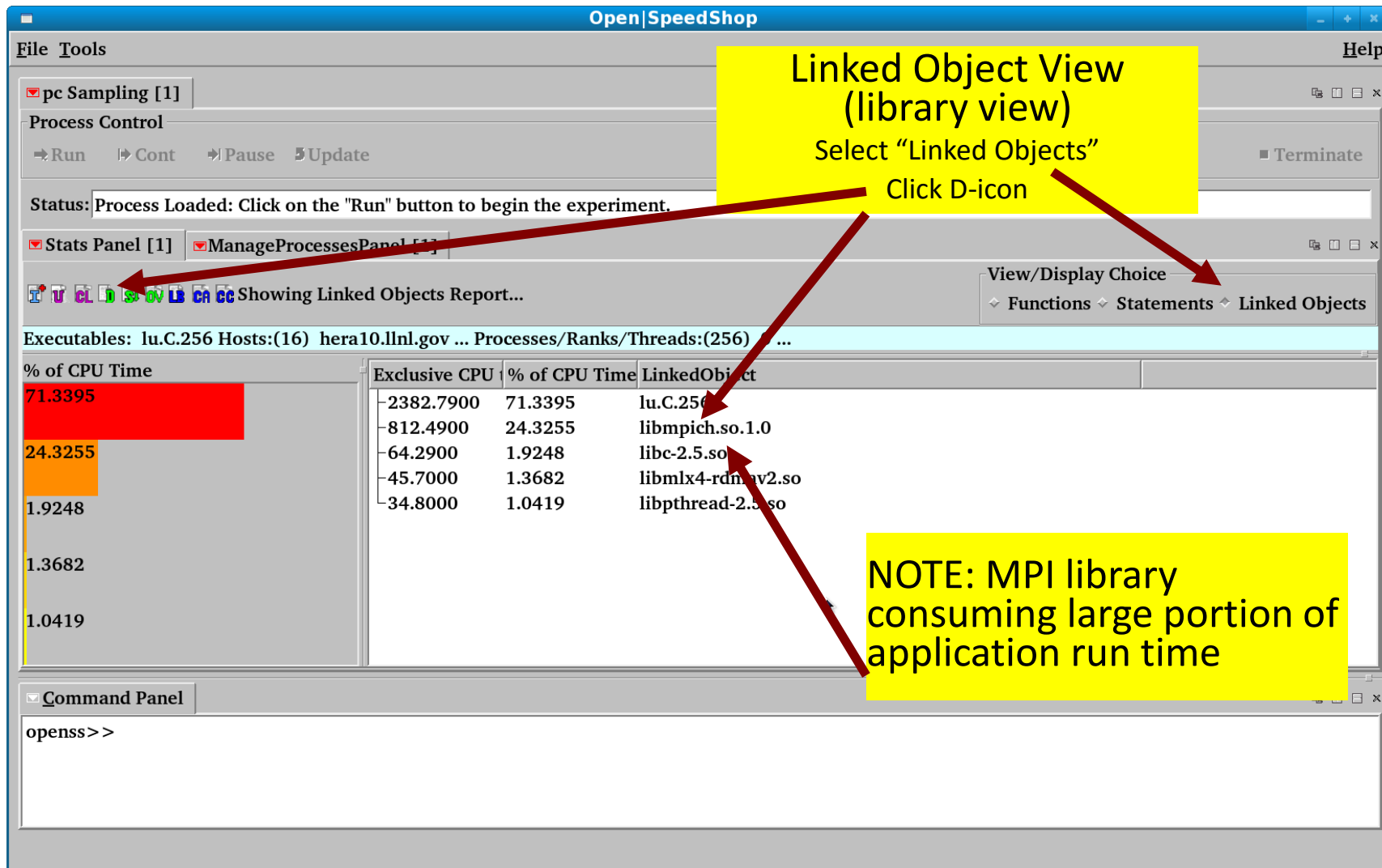
openss>>

MPI library showing up high in the list
Max time in rank 255

With load balance view we are looking for performance number out of norm of what is expected.

Large differences between min, max and/or average values.

❖ Default Aggregated View based on Linked Objects (libraries)



Open|SpeedShop

File Tools Help

pc Sampling [1]

Process Control

Run Cont Pause Update

Status: Process Loaded: Click on the "Run" button to begin the experiment.

Stats Panel [1] ManageProcessesPanel [1]

Showing Linked Objects Report...

View/Display Choice

Functions Statements Linked Objects

Executables: lu.C.256 Hosts:(16) hera10.llnl.gov ... Processes/Ranks/Threads:(256) ...

| % of CPU Time | Exclusive CPU | % of CPU Time | LinkedObject |
|---------------|---------------|---------------|-------------------|
| 71.3395 | 2382.7900 | 71.3395 | lu.C.256 |
| 24.3255 | 812.4900 | 24.3255 | libmpich.so.1.0 |
| 1.9248 | 64.2900 | 1.9248 | libc-2.5.so |
| 1.3682 | 45.7000 | 1.3682 | libmlx4-rdmav2.so |
| 1.0419 | 34.8000 | 1.0419 | libpthread-2.5.so |

Command Panel

openss>>

Linked Object View (library view)

Select "Linked Objects"

Click D-icon

NOTE: MPI library consuming large portion of application run time

MPI Tracing Results: Default View



❖ Default Aggregated MPI Experiment View

Information Icon Displays Experiment Metadata

Aggregated Results

Metadata for Experiment 1:
Application command:
Executables: smg2000
Experiment type: mpi
Host(s): hyperion583.llnl.gov hyperion584.llnl.gov hyperion585.llnl.gov hyperion586.llnl.gov hyperion587.llnl.gov hyperion588.llnl.gov hyperion589.llnl.gov h
Processes, Ranks or Threads: 0-511

| Minimum MPI Call Time(ms) | Maximum MPI Call Time(ms) | Average Time(ms) | Number of Calls | Function (defining location) |
|---------------------------|---------------------------|------------------|-----------------|---|
| 555.306000 | 1276.275000 | 755.289027 | 512 | PMPI_Init (libmonitor.so.0.0.0: pmpi.c,94) |
| 151.147000 | 167.504000 | 163.231094 | 512 | PMPI_Finalize (libmonitor.so.0.0.0: pmpi.c,223) |
| 0.152000 | 0.474000 | 0.334205 | 512 | MPI_Allgather (libmpich.so.1.0: allgather.c,73) |
| 0.043000 | 0.212000 | 0.133098 | 512 | MPI_Allgather (libmpich.so.1.0: allgather.c,70) |
| 0.031000 | 2.034000 | 1.312102 | 512 | MPI_Barrier (libmpich.so.1.0: barrier.c,56) |
| 0.013000 | 10.322000 | 0.717578 | 6144 | MPI_Allreduce (libmpich.so.1.0: allreduce.c,59) |
| 0.000001 | 611.617000 | 0.977852 | 4667648 | MPI_Waitall (libmpich.so.1.0: waitall.c,57) |
| 0.000001 | 0.600000 | 0.001156 | 5403936 | MPI_Isend (libmpich.so.1.0: isend.c,58) |
| 0.000001 | 0.069000 | 0.000665 | 5403936 | MPI_Irecv (libmpich.so.1.0: irecv.c,48) |

View Results: Show MPI Callstacks



Open|SpeedShop

File Tools Help

MPI [1]

Process Control

Run

Status: Process Loaded: Click on the "Run" button to begin the experiment.

Stats Panel [1] ManageProcessesPanel [1]

Showing CallTrees,FullStack Report:

Executables: smg2000 Host: localhost.localdomain Processes/Ranks/Threads:(2) 0 ...

| Exclusive MPI Call Time(ms) | % of Total | Number of Calls | Call Stack Function (defining location) |
|-----------------------------|------------|-----------------|---|
| 0.000000 | 0.000000 | | main (smg2000: smg2000.c,21) |
| 106.082304 | 12.510346 | 2 | @ 94 in PMPI_Init (libmonitor.so.0.0.0: pmpi.c,94) |
| 0.000000 | 0.000000 | | main (smg2000: smg2000.c,21) |
| 0.000000 | 0.000000 | | @ 49 in HYPRE_StructSMGSetup (smg2000: HYPRE_struct_smg.c,48) |
| 0.000000 | 0.000000 | | @ 335 in hypre_SMGSetup (smg2000: smg_setup.c,28) |
| 0.000000 | 0.000000 | | @ 405 in hypre_SMGRelaxSetup (smg2000: smg_relax.c,357) |
| 0.000000 | 0.000000 | | @ 613 in hypre_SMGRelaxSetupASol (smg2000: smg_relax.c,540) |
| 0.000000 | 0.000000 | | @ 335 in hypre_SMGSetup (smg2000: smg_setup.c,28) |
| 0.000000 | 0.000000 | | @ 405 in hypre_SMGRelaxSetup (smg2000: smg_relax.c,357) |
| 0.000000 | 0.000000 | | @ 619 in hypre_SMGRelaxSetupASol (smg2000: smg_relax.c,540) |
| 0.000000 | 0.000000 | | @ 549 in hypre_CyclicReductionSetup (smg2000: cyclic_reduction.c,4) |
| 0.000000 | 0.000000 | | @ 491 in hypre_StructCoarsen (smg2000: coarsen.c,139) |
| 24.321377 | 2.868234 | 600 | @ 39 in MPI_Waitall (libmpi.so.0.0.1: pwaitall.c,39) |

Command Panel

openss>>

Unique Call Paths View: Click C+ Icon

Unique Call Paths to MPI_Waitall and other MPI functions

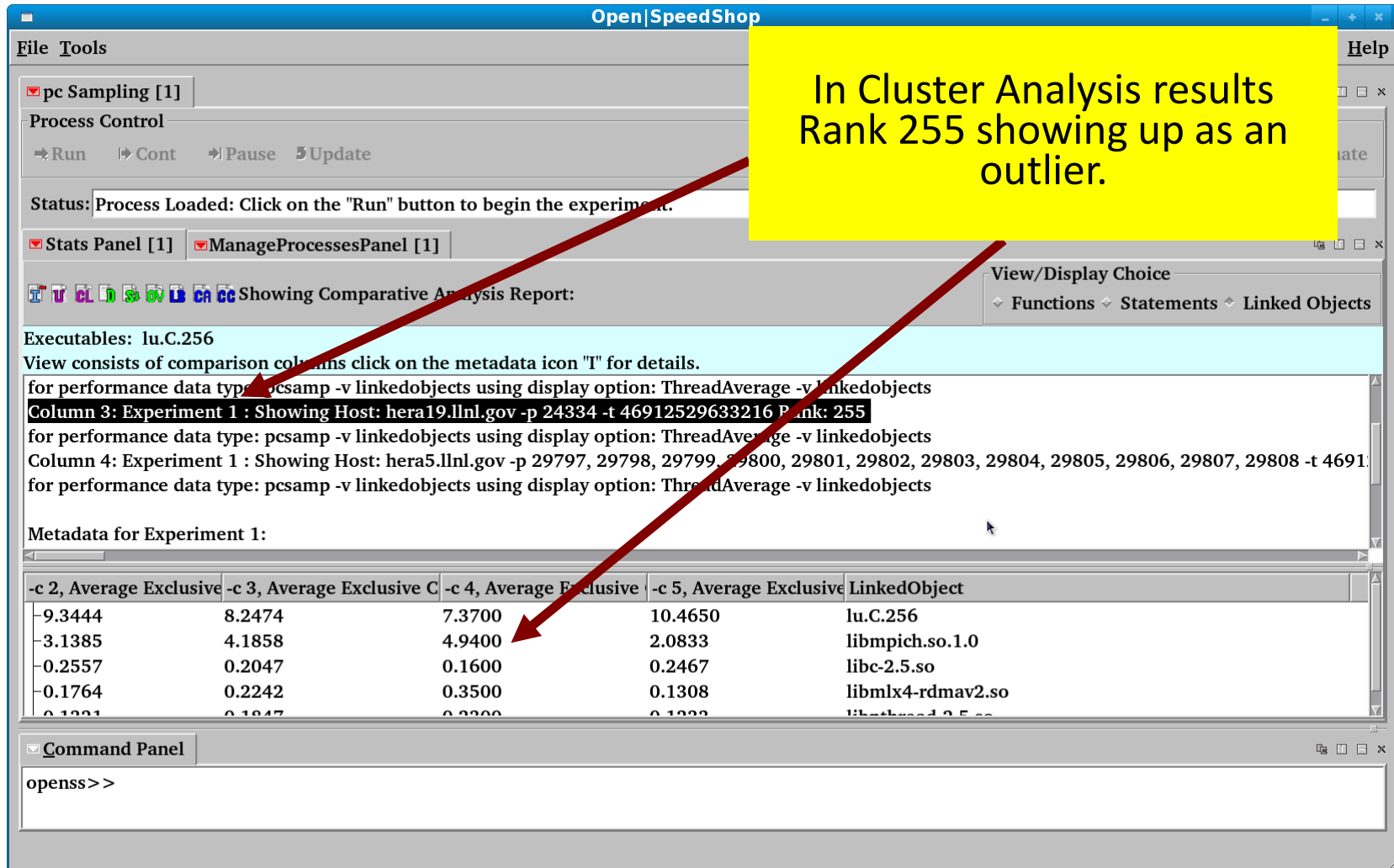
❖ Can use with pcsamp, usertime, hwc

- Will group like performing ranks/threads into groups
- Groups may identify outlier groups of ranks/threads
- Can examine the performance of a member of the outlier group
- Can compare that member with member of acceptable performing group

❖ Can use with mpi, mpit

- Same functionality as above w.r.t. cluster analysis
- But, now focuses on the performance of individual MPI_Functions.
- Key functions are MPI_Wait, MPI_WaitAll
- Can look at call paths to the key functions to analyze why they are being called to find performance issues

❖ Cluster Analysis View based on Linked Objects (libraries)



In Cluster Analysis results Rank 255 showing up as an outlier.

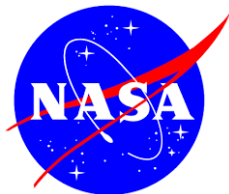
Executables: lu.C.256
View consists of comparison columns click on the metadata icon "T" for details.

for performance data type: pcsamp -v linkedobjects using display option: ThreadAverage -v linkedobjects
Column 3: Experiment 1 : Showing Host: hera19.llnl.gov -p 24334 -t 46912529633216 Rank: 255
for performance data type: pcsamp -v linkedobjects using display option: ThreadAverage -v linkedobjects
Column 4: Experiment 1 : Showing Host: hera5.llnl.gov -p 29797, 29798, 29799, 29800, 29801, 29802, 29803, 29804, 29805, 29806, 29807, 29808 -t 4691
for performance data type: pcsamp -v linkedobjects using display option: ThreadAverage -v linkedobjects

Metadata for Experiment 1:

| -c 2, Average Exclusive | -c 3, Average Exclusive C | -c 4, Average Exclusive | -c 5, Average Exclusive | LinkedObject |
|-------------------------|---------------------------|-------------------------|-------------------------|-------------------|
| -9.3444 | 8.2474 | 7.3700 | 10.4650 | lu.C.256 |
| -3.1385 | 4.1858 | 4.9400 | 2.0833 | libmpich.so.1.0 |
| -0.2557 | 0.2047 | 0.1600 | 0.2467 | libc-2.5.so |
| -0.1764 | 0.2242 | 0.3500 | 0.1308 | libmlx4-rdmav2.so |
| -0.1221 | 0.1847 | 0.2200 | 0.1222 | libpthread.so.0 |

Command Panel
openss>>



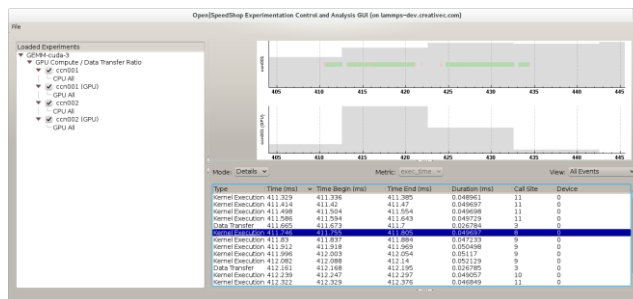
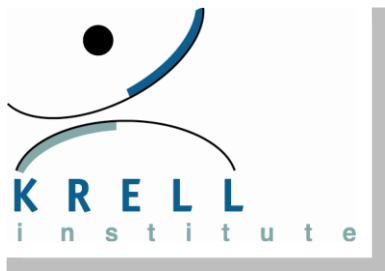
Open | SpeedShop™



Performance with Open/SpeedShop

NASA Open/SpeedShop Update/Training

Comparing Performance Data



❖ **Key functionality for any performance analysis**

- Absolute numbers often don't help
- Need some kind of baseline / number to compare against

❖ **Open | SpeedShop has support to line up profiles**

- Perform multiple experiments and create multiple databases
- Script to load all experiments and create multiple columns

❖ **Typical Example Comparisons**

- Between experiments to study improvements/changes
- Between ranks/threads to understand differences/outliers
- Before/after optimization
- Different configurations or inputs

❖ Convenience Script: **osscompare**

- Compares Open|SpeedShop up to 8 databases to each other
- Syntax: **osscompare “db1.openss,db2.openss,...”** [options]
 - osscompare man page has more details
- Produces side-by-side comparison listing
- Metric option parameter:
 - Compare based on: time, percent, a hwc counter, etc.
- Limit the number of lines by “rows=nn” option
- Specify the: **viewtype=[functions|statements|linkedobjects]**
 - View granularity: function, statement, or library level.
 - Function level is the default.
 - If statements option is specified:
 - Comparisons will be made by looking at the performance of each statement in all the databases that are specified.
 - Similar for libraries, if linkedobject is selected as the viewtype parameter.
- Options to write comparison output to comma separated list (csv) or text files

osscompare "smg2000-pcsamp.openss,smg2000-pcsamp-1.openss"

openss]: Legend: -c 2 represents smg2000-pcsamp.openss

[openss]: Legend: -c 4 represents smg2000-pcsamp-1.openss

-c 2, Exclusive CPU -c 4, Exclusive CPU Function (defining location)

time in seconds. time in seconds.

3.870000000 3.630000000 hypr_SMGResidual (smg2000: smg_residual.c,152)

2.610000000 2.860000000 hypr_CyclicReduction (smg2000: cyclic_reduction.c,757)

2.030000000 0.150000000 opal_progress (libopen-pal.so.0.0.0)

1.330000000 0.100000000 mca_btl_sm_component_progress (libmpi.so.0.0.2:
topo_unity_component.c,0)

0.280000000 0.210000000 hypr_SemiInterp (smg2000: semi_interp.c,126)

0.280000000 0.040000000 mca_pml_ob1_progress (libmpi.so.0.0.2:
topo_unity_component.c,0)



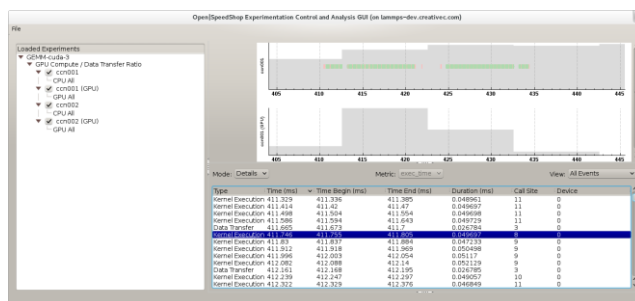
Open | SpeedShop™



Performance with Open | SpeedShop

NASA Open | SpeedShop Update/Training

Section 2: Recently added Functionality/Experiments



Section 1: Introduction to Open | SpeedShop tools

- How to use Open | SpeedShop to gather and display
- Overview of performance experiments
 - Sampling Experiments and Tracing Experiments
- How to compare performance data for different application runs

Section 2: New Functionality/Experiments

- Memory (ossmem) experiment
- OpenMP augmentation
- OMPTP (ossomptp) experiment
- POSIX threads (osspthread) experiment
- Lightweight experiments (ossiop, ossmpip)
- NVIDIA CUDA tracing experiment (osscuda)

Section 3: Roadmap / Future Plans

Supplemental Information

- Command Line Interface (CLI) tutorial and examples



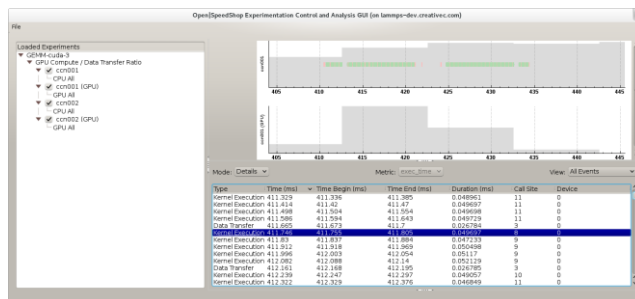
Open | SpeedShop™

Performance with Open | SpeedShop



NASA Open | SpeedShop Update/Training

Performance Analysis related to
application memory function activity



❖ **Supports sequential, mpi and threaded applications.**

- No instrumentation needed in application.
- Traces system calls via wrappers
 - malloc
 - calloc
 - realloc
 - free
 - memalign and posix_memalign

❖ **Provides metrics for**

- Timeline of events that set an new high-water mark.
- List of event allocations (with calling context) to leaks.
- Overview of all unique callpaths to traced memory calls that provides max and min allocation and count of calls on this path.

❖ **Example Usage**

- `ossmem "./lulesh2.0"`
- `ossmem "mpiexec_mpt -np 64 ./sweep3d.mpi"`

❖ **No GUI support at this time**

- Support planned via the new GUI, pending funding.

❖ **expview -vunique**

- Show times, call counts per path, min,max bytes allocation, total allocation to all unique paths to memory calls that the mem collector saw

❖ **expview -vleaked**

- Show function view of allocations that were not released while the mem collector was active

❖ **expview -vtrace,leaked**

- Will show a timeline of any allocation calls that were not released

❖ **expview -vfullstack,leaked**

- Display a full callpath to each unique leaked allocation

❖ **expview -v trace,highwater**

- Is a timeline of mem calls that set a new high-water
- The last entry is the allocation call that the set the high-water for the complete run
- Investigate the last calls in the timeline and look at allocations that have the largest allocation size (size1,size2,etc) if your application is consuming lots of system ram

❖ Shows the last 8 allocation events that set the high water mark

openss>>expview -vtrace,highwater

| Start Time(d:h:m:s) | Event Ids | Size Arg1 | Size Arg2 | Ptr Arg | Return Value | New Call Stack Function (defining location) Highwater |
|---|--------------|--------------|--------------|------------|--------------|--|
| *** trimmed all but the last 8 events of 61 *** | | | | | | |
| 2016/11/10 09:56:50.824 | 11877:0 | 2080 | 0 | 0x7760e0 | 19758988 | >>>>>>__GI___libc_malloc (libc-2.18.so) |
| 2016/11/10 09:56:50.826 | 11877:0 | 1728000 | 0 | 0x11783d0 | 21484908 | >>>>__GI___libc_malloc (libc-2.18.so) |
| 2016/11/10 09:56:50.827 | 11877:0 | 1728000 | 0 | 0x131e1e0 | 23212908 | >>>>__GI___libc_malloc (libc-2.18.so) |
| 2016/11/10 09:56:50.827 | 11877:0 | 1728000 | 0 | 0x14c3ff0 | 24940908 | >>>>__GI___libc_malloc (libc-2.18.so) |
| 2016/11/10 09:56:50.827 | 11877:0 | 2080 | 0 | 0x776a90 | 24942988 | >>>>>>__GI___libc_malloc (libc-2.18.so) |
| 2016/11/10 09:56:50.919 | 11877:0 | 1728000 | 0 | 0x1654030 | 25286604 | >>>>__GI___libc_malloc (libc-2.18.so) |
| 2016/11/10 09:56:50.919 | 11877:0 | 1728000 | 0 | 0x17f9e40 | 27014604 | >>>>__GI___libc_malloc (libc-2.18.so) |
| 2016/11/10 09:56:50.919 | 11877:0 | 2080 | 0 | 0xabc6a0 | 27016684 | >>>>>>__GI___libc_malloc (libc-2.18.so) |

- ❖ **The next slide shows** the default view of all unique memory calls seen while the mem collector was active. This is an overview of the memory activity. The default display is aggregated across all processes and threads. Ability to view specific processes or threads.
- ❖ **For all memory calls the following are displayed:**
 - The exclusive time and percent of exclusive time
 - The number of times this memory function was called.
 - The traced memory function name.
- ❖ **For allocation calls (e.g. malloc) the follow:**
 - The max and min allocation size seen.
 - The number of times the that max or min was seen are displayed.
 - The total allocation size of all allocations.

O|SS Memory Experiment (Unique Calls)



openss>>expview -vunique

| Exclusive (ms) | % of Total Time | Number of Calls | Min Request Count | Min Requested Bytes | Max Request Count | Max Requested Bytes | Total Bytes Requested | Function (defining location) |
|-------------------|-----------------------|-----------------------|-------------------------|---------------------------|-------------------------|---------------------------|-----------------------------|-----------------------------------|
| 0.024847 | 89.028629 | 1546 | 1 | 192 | 6 | 4096 | 6316416 | __GI___libc_malloc (libc-2.18.so) |
| 0.002371 | 8.495467 | 5 | | | | | | __GI___libc_free (libc-2.18.so) |
| 0.000369 | 1.322154 | 1 | 1 | 40 | 1 | 40 | 40 | __realloc (libc-2.18.so) |
| 0.000322 | 1.153750 | 3 | 1 | 368 | 1 | 368 | 1104 | __calloc (libc-2.18.so) |

NOTE: Number of Calls means the number of unique paths to the memory function call.

To see the paths use the CLI command: expview -vunique,fullstack

O|SS Memory Experiment (Leaked Calls)



In this example the sequential OpenMP version of lulesh was run under ossmem.

The initial run detected 69 potential leaks of memory.

Examining the calltrees using the cli command "**expview -vfullstack,leaked -mtot_bytes**" revealed that allocations from the Domain::Domain constructor where not later released in the Domain::~Domain destructor. After adding appropriate delete's in the destructor and rerunning ossmem, we observed a resolution of the leaks detected in the Domain class. The remaining leaks where minor and from system libraries.

Using the expstore command to load in the initial database and the database from the second run, we can use the expcompare cli command to see the improvements. Below, database -x1 shows the initial run and -x2 shows the results from the run with the changes to address the leaks detected in the Domain class.

```
openss>>expstore -f lulesh-mem-initial.openss
openss>>expstore -f lulesh-mem-improved.openss
openss>>expcompare -vleaked -mtot_bytes -mcalls -x1 -x2
```

| -x 1, Total Bytes Requested | -x 1, Number of Calls | -x 2, Total Bytes Requested | -x 2, Number of Calls | Function (defining location) |
|--------------------------------------|--------------------------------|--------------------------------------|--------------------------------|-----------------------------------|
| 10599396 | 69 | 3332 | 8 | __GI___libc_malloc (libc-2.17.so) |
| 72 | 1 | 72 | 1 | __realloc (libc-2.17.so) |

If timing permits:

- ❖ **Memory experiment related application exercise**

- More information provided at the tutorial

- ❖ **Memory exercises can be found in these directories:**

- \$HOME/exercises/matmul
 - \$HOME/exercises/lulesh2.0.3
 - \$HOME/exercises/lulesh2.0.3-fixed

- ❖ **Look for the README file for instructions.**



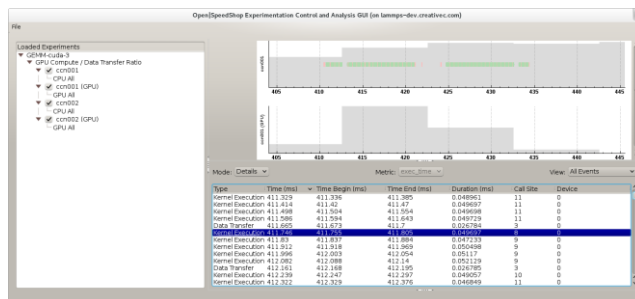
Open | SpeedShop™

Performance with Open/SpeedShop

NASA

NASA Open/SpeedShop Update/Training

Performance Analysis related to
application OpenMP activity



❖ O|SS augments the sampling experiments

- Applies the OMPT API callbacks for:
 - openmp thread idleness (waiting or work outside a parallel region)
 - openmp thread in barrier (within parallel region)
 - openmp thread waiting at a barrier (within parallel region)
- to samples taken in the OpenMP library that otherwise would be shown as in the Intel libiomp5 library
 - __kmp_barrier
 - __kmp_wait_sleep, etc.

❖ The user can see the aggregated sample time for idle, barrier, and wait_barrier.

❖ **With respect to the barrier symbols**

- barrier and wait_barrier occur within a parallel region and indicate time not doing work.
- idle means waiting for work outside the parallel region

❖ **Essentially these metrics as used in the O|SS sampling experiments to:**

- Inform the user the time a thread is idle and the time spent at a barrier (including waiting at a barrier).

❖ **The usertime experiment can give some context to where specific idle and barrier times are.**

- ❖ Using the usertime experiment on an OpenMP application can help to pinpoint where in the source the wait barrier time is coming from. For example:

- ❖ `openss>>expview`

| Exclusive CPU time in seconds. | Inclusive CPU time in seconds. | % of Total Exclusive CPU Time | Function (defining location) |
|---|---|--|---|
| 23.200000 | 23.200000 | 38.648263 | OMPT_THREAD_IDLE (usertime-collector-monitor-mrnet.so: collector.c,122) |
| 13.142857 | 13.142857 | 21.894336 | MAIN__.omp_fn.2 (stress_omp: stress_omp.f,179) |
| 12.885714 | 12.885714 | 21.465969 | MAIN__.omp_fn.5 (stress_omp: stress_omp.f,227) |
| 4.742857 | 4.742857 | 7.901000 | OMPT_THREAD_WAIT_BARRIER (usertime-collector-monitor-mrnet.so: collector.c,150) |
| 2.000000 | 11.771428 | 3.331747 | MAIN__ (stress_omp: stress_omp.f,1) |
| 1.257143 | 1.257143 | 2.094241 | __kernel_cof (libm-2.12.so: k_cof.c,45) |
| 1.085714 | 1.085714 | 1.808663 | __ieee754_rem_pio2f (libm-2.12.so: e_rem_pio2f.c,108) |

- ❖ Here we see the call path that points to the source lines that result in the thread waiting in the barrier.
- ❖ `openss>>expview -vcalltrees,fullstack -f OMPT_THREAD_WAIT_BARRIER usertime1`

| Exclusive CPU time in seconds. | Inclusive CPU time in seconds. | % of Total CPU Time | Call Stack Function (defining location) |
|---|---|---------------------------|---|
| | | | <code>_start (stress_omp)</code> |
| | | | <code>> @ 556 in __libc_start_main (libmonitor.so.0.0.0: main.c,541)</code> |
| | | | <code>>> __libc_start_main (libc-2.12.so)</code> |
| | | | <code>>>> @ 517 in monitor_main (libmonitor.so.0.0.0: main.c,492)</code> |
| | | | <code>>>>> main (stress_omp)</code> |
| | | | <code>>>>>> @ 227 in MAIN__ (stress_omp: stress_omp.f,1)</code> |
| | | | <code>>>>>>> @ 557 in __kmp_api_GOMP_parallel_end_10_alias (libiomp5.so: kmp_gsupport.c,490)</code> |
| | | | <code>>>>>>>> @ 2395 in __kmp_join_call (libiomp5.so: kmp_runtime.c,2325)</code> |
| | | | <code>>>>>>>>> @ 7114 in __kmp_internal_join (libiomp5.so: kmp_runtime.c,7093)</code> |
| | | | <code>>>>>>>>>> @ 1458 in __kmp_join_barrier(int) (libiomp5.so: kmp_barrier.cpp,1371)</code> |
| 1.742857 | 1.742857 | 2.903379 | <code>>>>>>>>>>> @ 150 in OMPT_THREAD_WAIT_BARRIER</code> |

(usertime-collector-monitor-mmet.so: collector.c,150)

O|SS OpenMP specific experiment information / example

Convenience script basic syntax:

ossomptp “executable”

Examples:

ossomptp “./openmp_stress < stress.input”

ossomptp “mpirun -np 16 ./lulesh2.0”

The following three CLI examples show the most important ways to view OMPTP experiment data.

- ❖ No GUI support at this time. Support planned via the new GUI, pending funding.

Default view shows the timing of the parallel regions, idle, barrier, and wait barrier as an aggregate across all threads

```
openss -cli -f ./matmult-omptp-0.openss
openss>>expview
```

| Exclusive times in seconds. | Inclusive times in seconds. | % of Total Exclusive | Function (defining location) |
|-----------------------------------|-----------------------------------|----------------------------|--|
| 44.638794 | 45.255843 | 93.499987 | compute._omp_fn.1 (matmult: matmult.c,68) |
| 1.744841 | 1.775104 | 3.654726 | compute_interchange._omp_fn.3 (matmult: matmult.c,118) |
| 0.701720 | 0.701726 | 1.469817 | compute_triangular._omp_fn.2 (matmult: matmult.c,95) |
| 0.652438 | 0.652438 | 1.366591 | IDLE (omptp-collector-monitor-mrnet.so: collector.c,573) |
| 0.004206 | 0.009359 | 0.008810 | initialize._omp_fn.0 (matmult: matmult.c,32) |
| 0.000032 | 0.000032 | 0.000068 | BARRIER (omptp-collector-monitor-mrnet.so: collector.c,587) |
| 0.000000 | 0.000000 | 0.000001 | WAIT_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,602) |

This example shows the comparison of exclusive time across all threads for the parallel regions, idle, barrier, and wait barrier

```
openss>>expcompare -mtime -t0:4
```

-t 0, -t 2, -t 3, -t 4, Function (defining location)

**Exclusive Exclusive Exclusive Exclusive
times in times in times in times in
seconds. seconds. seconds. seconds.**

| | | | | |
|-----------|-----------|-----------|-----------|--|
| 11.313892 | 11.081346 | 11.313889 | 10.929668 | compute._omp_fn.1 (matmult: matmult.c,68) |
| 0.443713 | 0.430553 | 0.429635 | 0.440940 | compute_interchange._omp_fn.3 (matmult: matmult.c,118) |
| 0.253632 | 0.213238 | 0.164875 | 0.069975 | compute_triangular._omp_fn.2 (matmult: matmult.c,95) |
| 0.001047 | 0.001100 | 0.001095 | 0.000964 | initialize._omp_fn.0 (matmult: matmult.c,32) |
| 0.000008 | 0.000008 | 0.000006 | 0.000010 | BARRIER (omptp-collector-monitor-mrnet.so: collector.c,587) |
| 0.000000 | 0.000000 | 0.000000 | 0.000000 | WAIT_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,602) |
| 0.000000 | 0.247592 | 0.015956 | 0.388890 | IDLE (omptp-collector-monitor-mrnet.so: collector.c,573) |

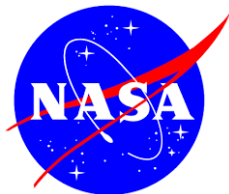
This example shows the load balance of time across all threads for the parallel regions, idle, barrier, and wait barrier

```
openss>>expview -mloadbalance
```

| Max Exclusive Time Across OpenMP ThreadIds(s) | OpenMP ThreadId of Max OpenMP ThreadIds(s) | Min Exclusive Time Across OpenMP ThreadIds(s) | OpenMP ThreadId of Min OpenMP ThreadIds(s) | Average Exclusive Time Across ThreadIds(s) | Function (defining location) |
|---|---|---|---|---|---|
| 11.313892 | 0 | 10.929668 | 4 | 11.159699 | compute._omp_fn.1 (matmult: matmult.c,68) |
| 0.443713 | 0 | 0.429635 | 3 | 0.436210 | compute_interchange._omp_fn.3 (matmult: matmult.c,118) |
| 0.388890 | 4 | 0.015956 | 3 | 0.217479 | IDLE (omptp-collector-monitor-mrnet.so: collector.c,573) |
| 0.253632 | 0 | 0.069975 | 4 | 0.175430 | compute_triangular._omp_fn.2 (matmult: matmult.c,95) |
| 0.001100 | 2 | 0.000964 | 4 | 0.001052 | initialize._omp_fn.0 (matmult: matmult.c,32) |
| 0.000010 | 4 | 0.000006 | 3 | 0.000008 | BARRIER (omptp-collector-monitor-mrnet.so: collector.c,587) |
| 0.000000 | 0 | 0.000000 | 0 | 0.000000 | WAIT_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,602) |

If timing permits:

- ❖ **OpenMP specific experiment application exercise**
- ❖ **OpenMP profiling exercises can be found in these directories:**
 - `$HOME/exercises/matmul`
 - `$HOME/exercises/hybrid_lulesh2.0.3`
 - `$HOME/exercises/lulesh2.0.3`



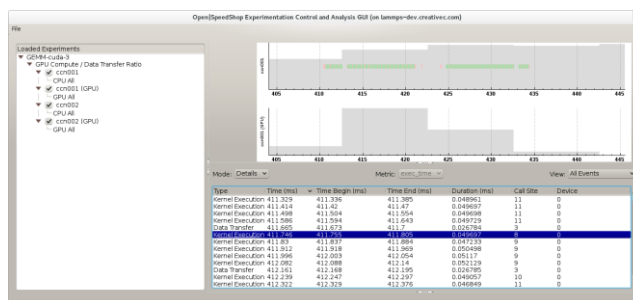
Open | SpeedShop™

Performance with Open/SpeedShop



NASA Open/SpeedShop Update/Training

Performance Analysis related to
application POSIX thread activity



***pthreads* experiment was created using the CBTF infrastructure**

- ❖ Gives opportunity to filter the POSIX thread performance information to reduce and mine the important/worthwhile information while the data is transferring to the client tool
 - **Discussion Topic: What is that worthwhile information?**
 - **Ideas:**
 - Report statistics about pthread wait
 - Report OMP blocking times
 - Attribute information to proper threads
 - Thread numbering improvements
 - Use a shorter alias number for the long POSIX pthread numbers
 - Report synchronization overhead mapped to proper thread
- ❖ Slides that follow show what the tool provides currently

O|SS pthreads experiment information and example

Convenience script basic syntax:

osspthread “executable” [default | <list of POSIX func>]

➤ Parameters

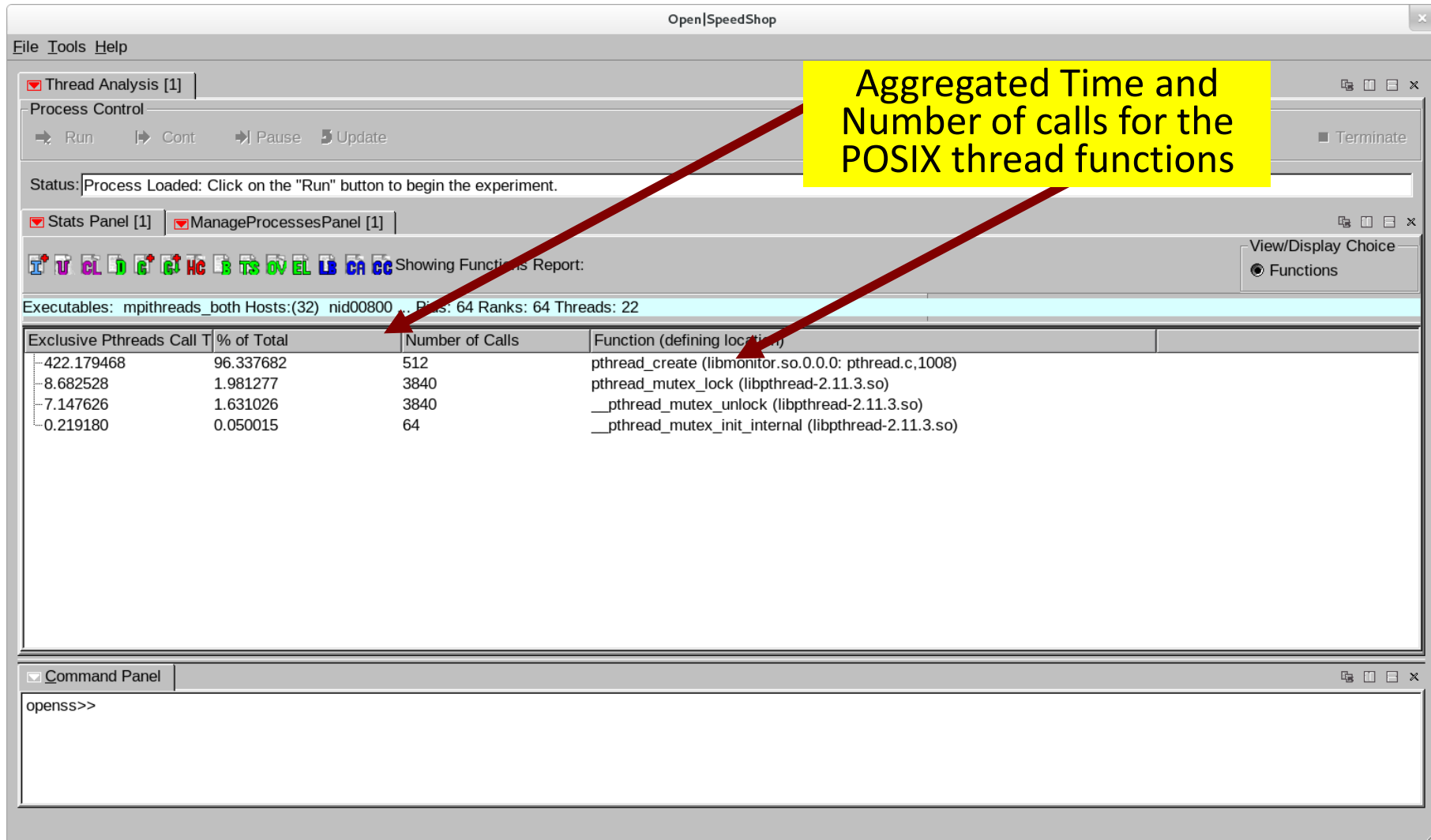
- POSIX thread function list to sample(default is all)
- pthread_create, pthread_mutex_init, pthread_mutex_destroy, pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock, pthread_cond_init, pthread_cond_destroy, pthread_cond_signal, pthread_cond_broadcast, pthread_cond_wait, pthread_cond_timedwait

Examples:

osspthread “aprun -n 64 -d 8 ./mpithreads_both”

osspthread “mpirun -np 256 sweep3d.mpi” pthread_mutex_lock

OSS/CBTF pthreads experiment default GUI view



Thread Analysis [1]

Process Control

Run Cont Pause Update

Status: Process Loaded: Click on the "Run" button to begin the experiment.

Stats Panel [1] ManageProcessesPanel [1]

Showing Functions Report:

Executables: mpithreads_both Hosts:(32) nid00800 ... Ranks: 64 Threads: 22

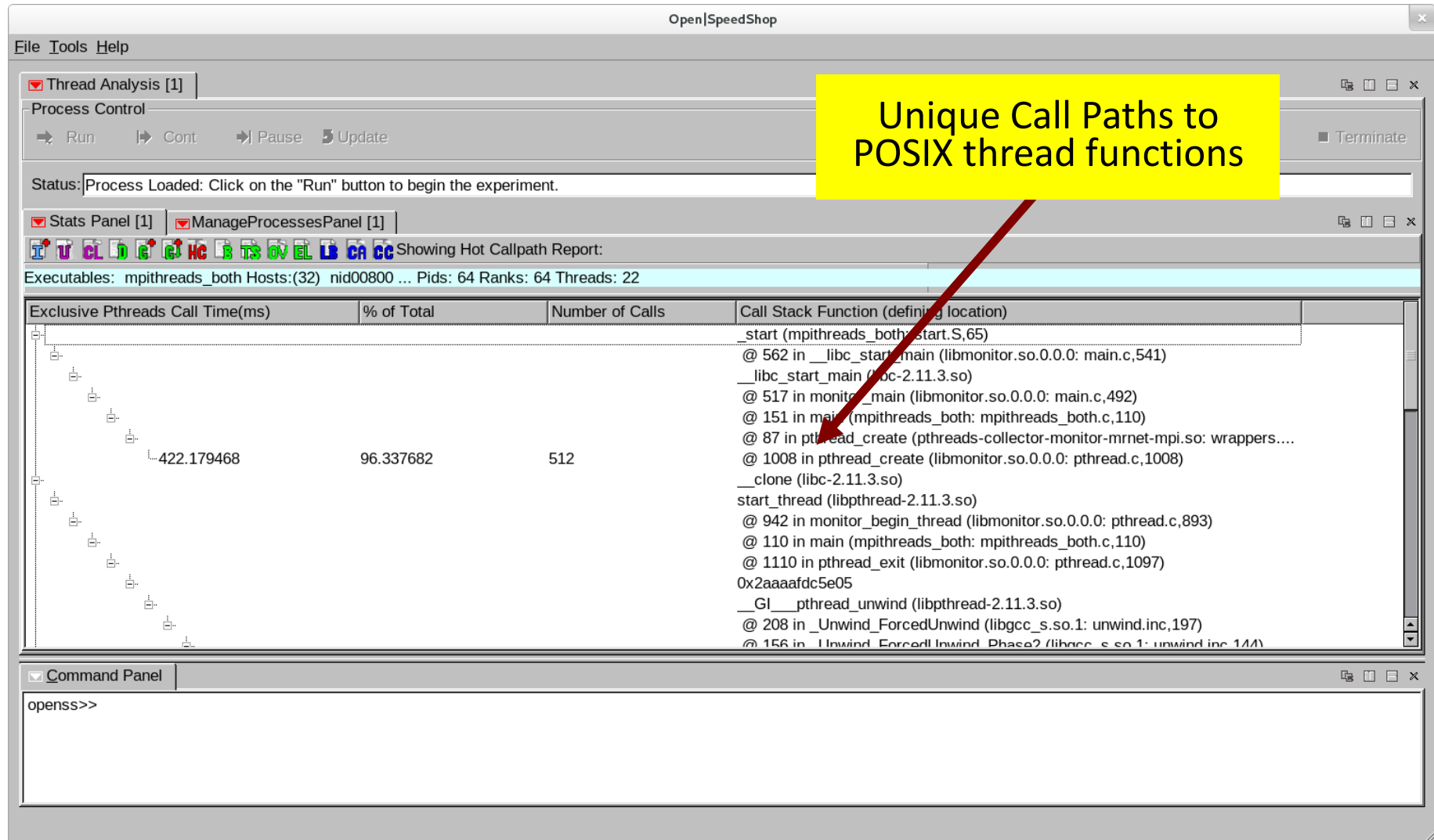
| Exclusive Pthreads Call T | % of Total | Number of Calls | Function (defining location) |
|---------------------------|------------|-----------------|--|
| -422.179468 | 96.337682 | 512 | pthread_create (libmonitor.so.0.0.0: pthread.c,1008) |
| -8.682528 | 1.981277 | 3840 | pthread_mutex_lock (libpthread-2.11.3.so) |
| -7.147626 | 1.631026 | 3840 | __pthread_mutex_unlock (libpthread-2.11.3.so) |
| -0.219180 | 0.050015 | 64 | __pthread_mutex_init_internal (libpthread-2.11.3.so) |

Command Panel

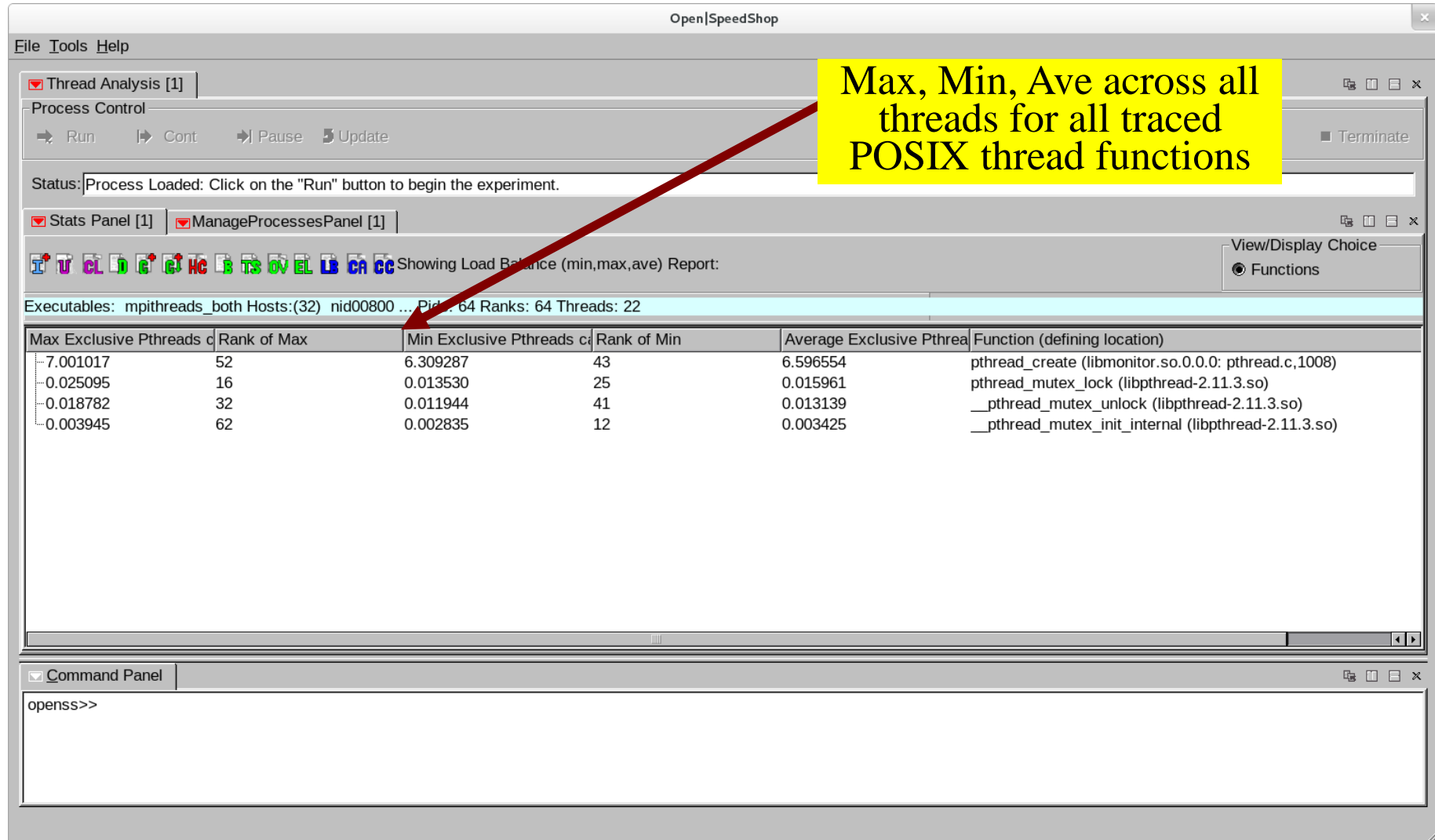
openss>>

Aggregated Time and Number of calls for the POSIX thread functions

OSS/CBTF pthreads experiment callpath GUI view



OSS/CBTF pthreads experiment loadbalance GUI view



Max, Min, Ave across all threads for all traced POSIX thread functions

Thread Analysis [1]

Process Control

Run Cont Pause Update

Status: Process Loaded: Click on the "Run" button to begin the experiment.

Stats Panel [1] ManageProcessesPanel [1]

Showing Load Balance (min,max,ave) Report:

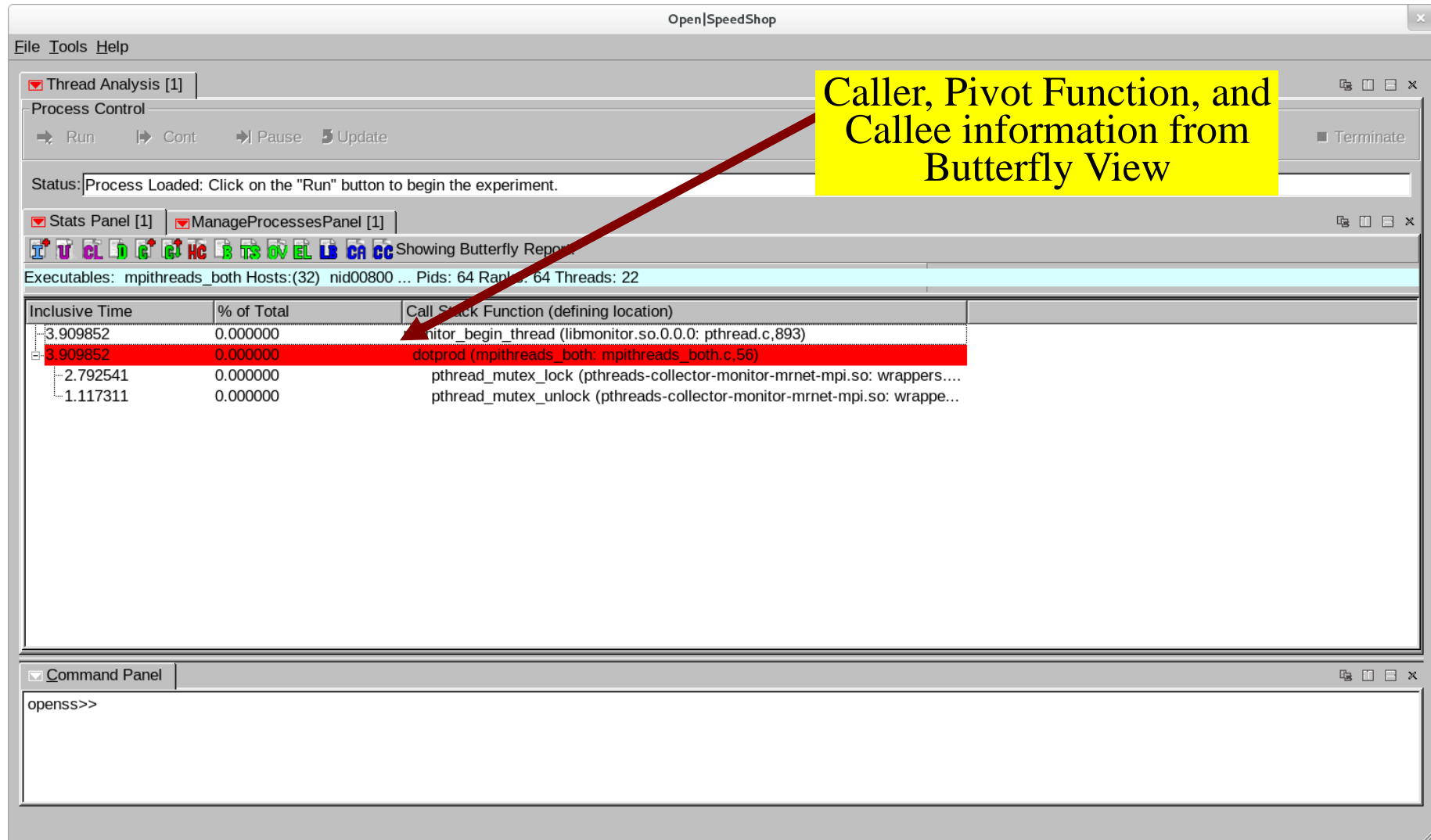
Executables: mpthreads_both Hosts:(32) nid00800 ... Pid: 64 Ranks: 64 Threads: 22

| Max Exclusive Pthreads c | Rank of Max | Min Exclusive Pthreads c | Rank of Min | Average Exclusive Pthrea | Function (defining location) |
|--------------------------|-------------|--------------------------|-------------|--------------------------|--|
| 7.001017 | 52 | 6.309287 | 43 | 6.596554 | pthread_create (libmonitor.so.0.0.0: pthread.c,1008) |
| 0.025095 | 16 | 0.013530 | 25 | 0.015961 | pthread_mutex_lock (libpthread-2.11.3.so) |
| 0.018782 | 32 | 0.011944 | 41 | 0.013139 | __pthread_mutex_unlock (libpthread-2.11.3.so) |
| 0.003945 | 62 | 0.002835 | 12 | 0.003425 | __pthread_mutex_init_internal (libpthread-2.11.3.so) |

Command Panel

openss>>

OSS/CBTF pthreads experiment butterfly GUI view



Open|SpeedShop

File Tools Help

Thread Analysis [1]

Process Control

Run Cont Pause Update

Status: Process Loaded: Click on the "Run" button to begin the experiment.

Stats Panel [1] ManageProcessesPanel [1]

Showing Butterfly Report

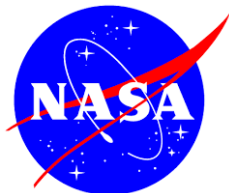
Executables: mpithreads_both Hosts:(32) nid00800 ... Pids: 64 Rapi... 64 Threads: 22

| Inclusive Time | % of Total | Call Stack Function (defining location) |
|----------------|------------|---|
| 3.909852 | 0.000000 | monitor_begin_thread (libmonitor.so.0.0.0: pthread.c,893) |
| 3.909852 | 0.000000 | dotprod (mpithreads_both: mpithreads_both.c,56) |
| -2.792541 | 0.000000 | pthread_mutex_lock (pthreads-collector-monitor-mrnet-mpi.so: wrappers.... |
| -1.117311 | 0.000000 | pthread_mutex_unlock (pthreads-collector-monitor-mrnet-mpi.so: wrappe... |

Command Panel

openss>>

Caller, Pivot Function, and Callee information from Butterfly View



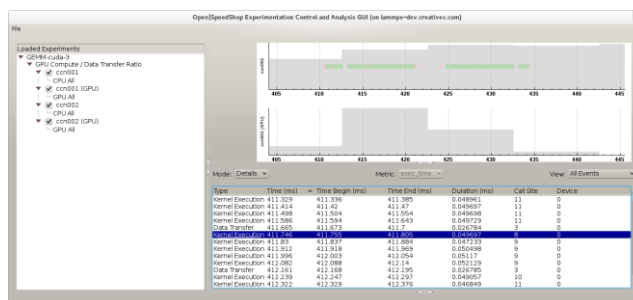
Open | SpeedShop™

Performance with Open/SpeedShop



NASA Open/SpeedShop Update/Training

Lightweight I/O and MPI



Lightweight Experiments

- ❖ **iop** – Gather I/O information like the io experiment, but do not save the information about each individual I/O call.
- ❖ **mpip** - Gather MPI information like the io experiment, but do not save the information about each individual MPI call.
- ❖ **Experiments still give a good overview of I/O and MPI, but reduce the sizes of the Open | SpeedShop database created.**
- ❖ **Size comparison:**
 - 668K smg2000-mpip-0.openss
 - ossmpip "mpirun -np 4 ./smg2000 -n 10 10 10"
 - 5.0M smg2000-mpi-0.openss
 - ossmpi "mpirun -np 4 ./smg2000 -n 10 10 10"
 - 12M smg2000-mpit-0.openss
 - ossmpit "mpirun -np 4 ./smg2000 -n 10 10 10"
 - 60K smg2000-mpit-1.openss
 - Gathered data for only the MPI collective mpi category.
 - ossmpit "mpirun -np 4 ./smg2000 -n 10 10 10" collective_com



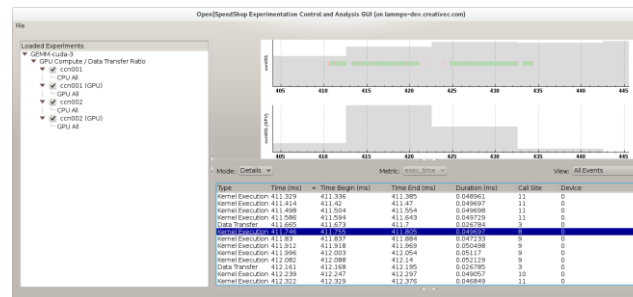
Open | SpeedShop™

Performance with Open/SpeedShop



NASA Open/SpeedShop Update/Training

NVIDIA CUDA Performance Analysis



What performance info does O|SS provide?

❖ For GPGPU O|SS reports information to help understand:

- Time spent in the GPU device
- Cost and size of data transferred to/from the GPU
- Balance of CPU versus GPU utilization
- Transfer of data between the host and device memory versus the execution of computational kernels
- Performance of the internal computational kernel code running on the GPU device

❖ O|SS is able to monitor CUDA scientific libraries because it operates on application binaries.

❖ Support for CUDA based applications is provided by tracing actual CUDA events

❖ OpenACC support is conditional on the CUDA RT.

Usage:

osscuda "executable" [extra_args]

Where "executable" is defined as the command that you normally use to execute your program but placed in quotes.

Example: **osscuda "mpiexec_mpt -np 8 ./Triad" [extra_args]**

The optional "extra_args" are defined as follows:

The following arguments control the periodic sampling of both CPU and GPU hardware performance counters performed by the cuda collector:

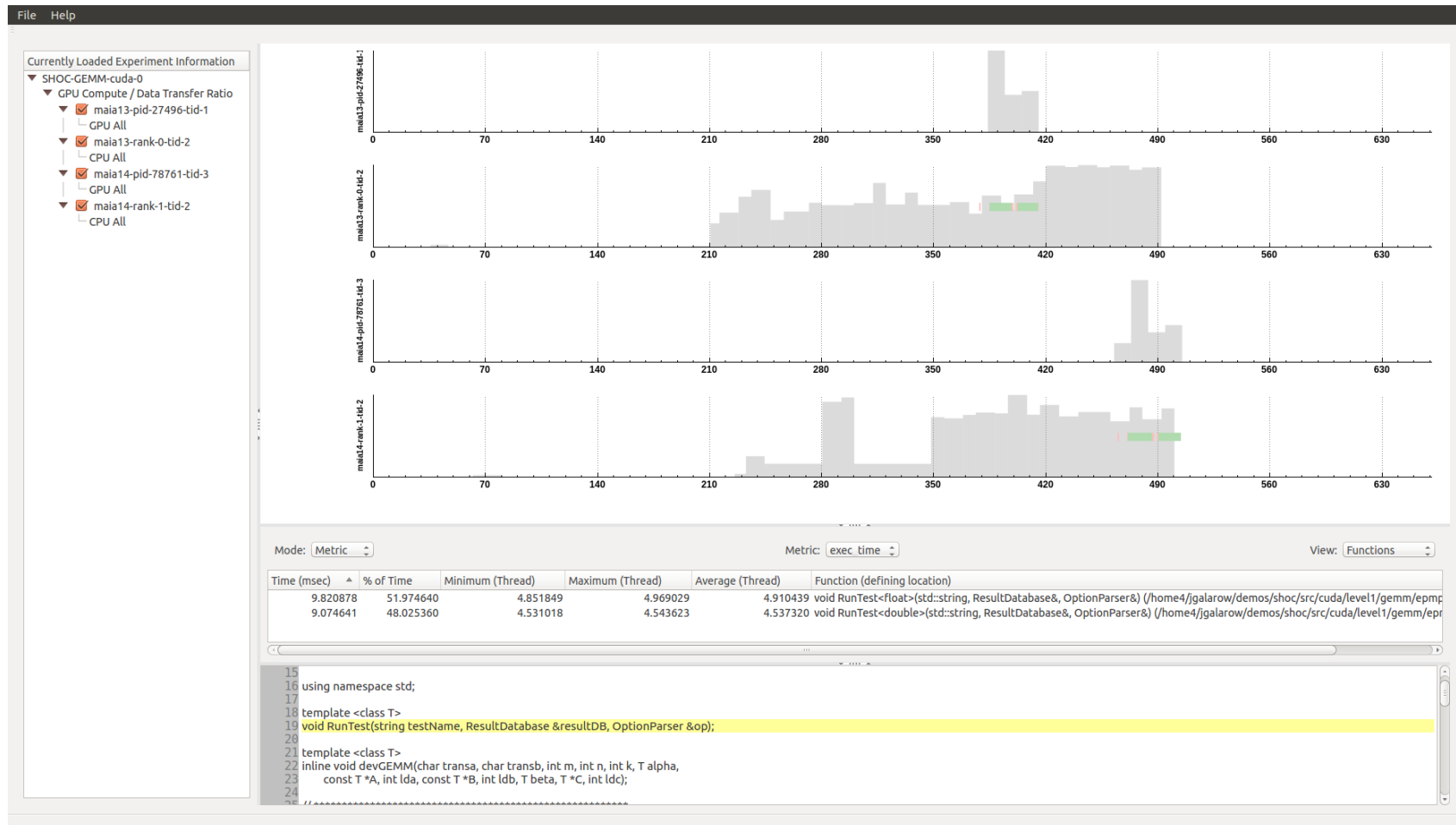
- "all" - Periodically sample all instructions.
- "branches" - Periodically sample branch instructions.
- "integer" - Periodically sample integer instructions.
- "single" - Periodically sample single-precision float instructions.
- "double" - Periodically sample double-precision float instructions.
- "memory" - Periodically sample load/store instructions.

- "low" - Periodically sample the requested instructions every 100 ms.
- "default" - Periodically sample the requested instructions every 10 ms.
- "high" - Periodically sample the requested instructions every 1 ms.

CUDA GUI View: Default CUDA view



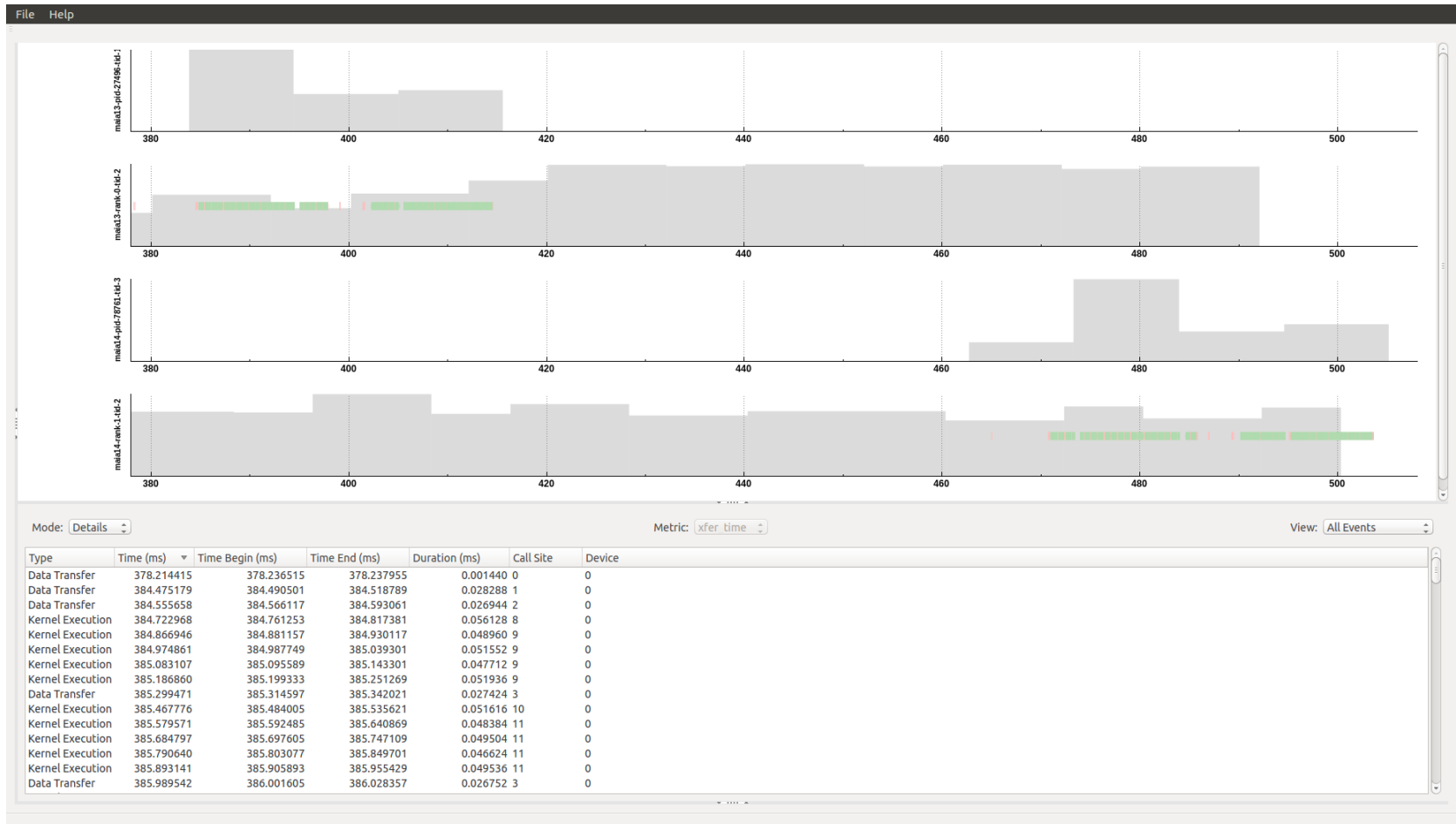
Note: The left pane shows the executable and the nodes it ran on. In future, will effect views.
Internal GPU activity is shown in ccn0001 (GPU All) graphic (shaded area)
Red boxes indicate data transfers, Green boxes indication GPU kernel executions
Source panel displays source for metrics clicked on in the Metric pane.



CUDA GUI View: All Events Trace



Note: This is the “All Events” Details View which shows the chronological list of CUDA kernel executions and data transfers. Here the Experiment Panel (the left side panel) has been completely collapsed to maximize the width of the right-side panels.



Section 1: Introduction to Open | SpeedShop tools

- How to use Open | SpeedShop to gather and display
- Overview of performance experiments
 - Sampling Experiments and Tracing Experiments
- How to compare performance data for different application runs

Section 2: New Functionality/Experiments

- Memory (ossmem) experiment
- OpenMP augmentation
- OMPTP (ossomptp) experiment
- POSIX threads (osspthread) experiment
- Lightweight experiments (ossiop, ossmpip)

Section 3: Roadmap / Future Plans

Supplemental Information

- Command Line Interface (CLI) tutorial and examples



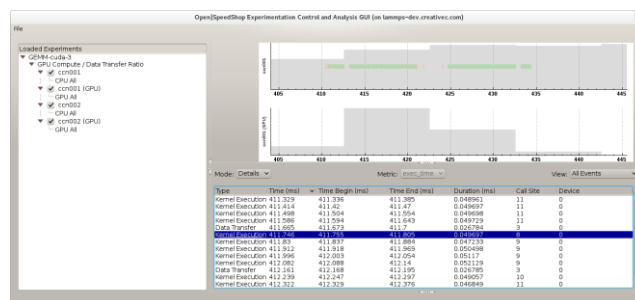
Open | SpeedShop™

NASA

Performance with Open | SpeedShop

NASA Open | SpeedShop Update/Training

Section 3 Road Map / Future Work



❖ **Component Based Tool Framework (CBTF)**

- **New version of O|SS uses tree based network (MRNet)**
 - Transfer data over the network, does not write files like the offline version
 - Allows the possibility of data reduction (in parallel) as the data is streamed up the tree
- **Six new experiments implemented in this version**
 - Lightweight I/O profiling (iop)
 - Lightweight MPI profiling (mpip)
 - Threading experiments (pthreads)
 - Memory usage analysis (mem)
 - GPU/Accelerator support (cuda)
 - OpenMP specific support (omptp)

❖ New features and improvements in O|SS

- **OpenMP idle/wait time augmentation to sampling experiments**
- **Spack build support for clusters – not Cray yet**
- **Conversion to cmake builds for O|SS, CBTF from GNU auto tools.**
- **Support for offline like capability in the O|SS CBTF version**
 - Use osspcsamp --offline “how you run your application normally”
 - Same for other experiments: ossusertime, osshwc, etc..
- **Fix in Qt3 GUI for better support of function related views when function name is STL or C++ namespace based.**
- **Major improvements to NVIDIA CUDA GPU experiment**
 - Initial new GUI creation
 - Improved performance data collection
 - Improved command line interface (CLI) views
- **ARM, Power8 support**
- **Tracing of MPI asynchronous non-blocking functions in the MPI experiments.**

New functionality being worked on now or planned

❖ **Creation of an Overview experiment**

- Give users an overview of the performance of their application
- Include information in a lightweight manner
- Include MPI, I/O, hardware counters, PC sampling, other
- May not create a database? Still in the planning stages
- Task for Tri-labs listed in development contract

❖ **Continue improving Intel MIC (KNL) support**

❖ **Filtering (data reduction, analysis) in the MRNet communication nodes**

- Faster views as data is mined in parallel

❖ **Investigate performance analysis by phases and iteration of the phase, perhaps using LLNL caliper project.**

❖ **Spack based OpenSpeedShop builds for Cray platform**

❖ **In discussion: replacement/upgrade for mpiotf experiment to write OTF-2 instead of OTF. OTF == Open Trace Format**

Pleades platform:

❖ **module use /home4/jgalarow/privatemodules**

❖ **Module names:**

- module load openspeedshop (defaults to mpt)
- module load openspeedshop.mpt
- module load openspeedshop.intelmpi
- module load openspeedshop.mvapich2
- module load openspeedshop.openmpi

KNL cluster platform:

❖ **module use /u/jgalarow/privatemodules**

❖ **Module names:**

- module load openspeedshop (defaults to mpt)
- module load openspeedshop.mpt
- module load openspeedshop.intelmpi

For mpi* experiments use the module file that corresponds to the MPI implementation your application was built with.

- ❖ **Current version: 2.3.1 has been released**
- ❖ **Open | SpeedShop Website**
 - <http://www.openspeedshop.org/>
- ❖ **Open | SpeedShop help and bug reporting**
 - [Direct email: oss-contact@openspeedshop.org](mailto:oss-contact@openspeedshop.org)
 - [Forum/Group: oss-questions@openspeedshop.org](mailto:oss-questions@openspeedshop.org)
- ❖ **Feedback**
 - Bug tracking available from website
 - Feel free to contact presenters directly
 - Support contracts and onsite training available

❖ Build and Installation Instructions

- <http://www.openspeedshop.org/documentation>
 - Look for: Open | SpeedShop Version 2.3 Build/Install Guide

❖ Open | SpeedShop User Guide Documentation

- <http://www.openspeedshop.org/documentation>
 - Look for Open | SpeedShop Version 2.3 Users Guide

❖ Man pages: OpenSpeedShop, osspcsamp, ossmpi, ...

❖ Quick start guide downloadable from web site

- <http://www.openspeedshop.org>
- Click on “Download Quick Start Guide” button

Section 1: Introduction to Open | SpeedShop tools

- How to use Open | SpeedShop to gather and display
- Overview of performance experiments
 - Sampling Experiments and Tracing Experiments
- How to compare performance data for different application runs

Section 2: New Functionality/Experiments

- Memory (ossmem) experiment
- OpenMP augmentation
- OMPTP (ossomptp) experiment
- POSIX threads (osspthread) experiment
- Lightweight experiments (ossiop, ossmpip)

Section 3: Roadmap / Future Plans

Supplemental Information

- Command Line Interface (CLI) tutorial and examples



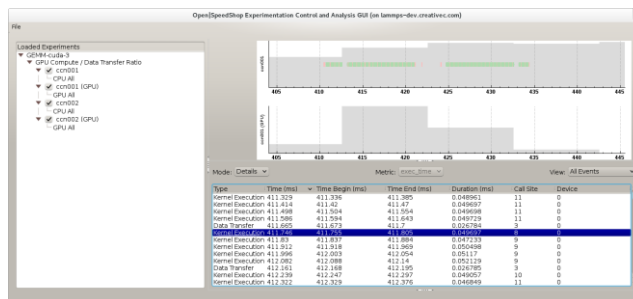
Open | SpeedShop™

Performance with Open/SpeedShop

NASA

NASA Open/SpeedShop Update/Training

Supplemental: 1 Command Line Interface Usage



❖ Command Line Interface Features

- “gdb” like tool for performance data creation and viewing
- Same functional capabilities the graphical user interface (GUI)
 - Exception: GUI can focus on the source line corresponding to statistics
- List metadata about your application and the OSS experiment
- Create experiments and run them
- Launch the GUI from the CLI via the “opengui” command
- View performance data from a database file
 - openss -cli -f <database filename> to launch
 - expview – key command with many options
 - list – list many items such as source, object files, metrics to view
 - expcompare – Compare ranks, threads, processes to each other, more...
 - cviewcluster – Creates groups of like performing entities to outliers.
 - cview – Output the columns of data representing groups created by cviewcluster.
 - cviewinfo – Output what ranks, threads, etc., are in each cview group

❖ Command Line Interface Features (additional)

- Format the performance information view to csv
 - `expview -F csv`
- Selectively only view, compare, analyze by rank, thread, process
 - `-r <rank number>` or rank list or rank ranges
 - `-t <thread number>` or thread list or thread ranges
 - `-p <process number>` or process list or process ranges
- Selectively only view, compare, analyze by specific metrics
 - `-m <metric name>` or list of metrics
 - Example metrics for sampling: percent, time
 - Example metrics for tracing: time, count, percent
- Selectively only view by specific view type options
 - `-v <view type>` or list of view types
 - Example view types: functions, statements, loops, linked objects
 - Example metrics for tracing: time, count, percent

Command Line Interface Examples

❖ **openss –cli –f <database file name>**

❖ **Commands to get started**

➤ **expstatus**

- Gives the metadata information about the experiment

➤ **expview**

- Displays the default view for the experiment
- Use expview <experiment type>nn to see only nn lines of output
 - expview pcsamp20 shows only the top 20 time taking functions
- -v functions : displays data based on function level granularity
- -v statements : displays data based on statement level granularity
- -v linkedobjects : displays data based on library level granularity
- -v loops : displays data based on loop level granularity
- -v calltrees : displays call paths combining like paths
- -v calltrees,fullstack : displays all unique call paths individually
- -m loadbalance : displays the min, max, average values across ranks, ...

Command Line Interface Examples

❖ Openss -cli -f <database file name>

❖ Commands to get started

- expview (continued from previous page)
 - -v trace : for tracing experiments, display chronological list of events
 - -m <metric> : only display the metric(s) provided via the -m option
 - Where metric can be: time, percent, a hardware counter, (see list -v metrics)
 - -r <rank or rank list> : only display data for that rank or ranks
 - -t < thread id or list of thread ids> : only display data for that thread (s)
 - -h < host id or list of host ids> : only display data for that host or hosts
 - -F csv : display performance data in a comma separated list
- expcompare : compare data within the same experiment
 - -r 1 -r 2 -m time : compare rank 1 to rank 2 for metric equal time
 - -h host1 -h host2 : compare host 1 to host 2 for the default metric

Command Line Interface Examples

❖ **openss –cli –f <database file name>**

❖ **Commands to get started**

➤ list

- -v metrics : display the data types (metric) that can be displayed via –m
- -v src : display source files associated with experiment
- -v obj : display object files associated with experiment
- -v ranks : display ranks associated with experiment
- -v hosts : display machines associated with experiment
- -v exp : display the experiment numbers that are currently loaded
- -v savedviews : display the commands that are cached in the database

openss -cli -f smg2000-hwcsamp-1.openss

View the default report for this hwcsamp experiment

openss>>[openss]: The restored experiment identifier is: -x 1

openss>>**expview**

| Exclusive CPU time in seconds. | % of CPU Time | PAPI_TOT_CYC | PAPI_FP_OPS | Function (defining location) |
|--|---------------|--------------|-------------|--------------------------------------|
| 3.920000000 smg_residual.c,152) | 44.697833523 | 11772604888 | 1198486900 | hypre_SMGResidual (smg2000: |
| 2.510000000 cyclic_reduction.c,757) | 28.620296465 | 7478131309 | 812850606 | hypre_CyclicReduction (smg2000: |
| 0.310000000 | 3.534777651 | 915610917 | 48863259 | opal_progress (libopen-pal.so.0.0.0) |
| 0.300000000 semi_restrict.c,125) | 3.420752566 | 910260309 | 100529525 | hypre_SemiRestrict (smg2000: |
| 0.290000000 (libmpi.so.0.0.2) | 3.306727480 | 874155835 | 48509938 | mca_btl_sm_component_progress |

Viewing hwcsamp data in CLI



View the linked object (library) view for this Hardware Counter Sampling experiment

openss>>**expview -v linkedobjects**

| Exclusive CPU time in seconds. | % of CPU Time | PAPI_TOT_CYC | PAPI_FP_OPS | LinkedObject |
|-----------------------------------|---------------|--------------|-------------|----------------------|
| 7.710000000 | 87.315968290 | 22748513124 | 2396367480 | smg2000 |
| 0.610000000 | 6.908267271 | 1789631493 | 126423208 | libmpi.so.0.0.2 |
| 0.310000000 | 3.510758777 | 915610917 | 48863259 | libopen-pal.so.0.0.0 |
| 0.200000000 | 2.265005663 | 521249939 | 46127342 | libc-2.10.2.so |
| 8.830000000 | 100.000000000 | 25975005473 | 2617781289 | Report Summary |

openss>>

Viewing I/O (iot) data in CLI



View the list of metrics (types of performance information) for this I/O experiment

```
openss>>list -v metrics
```

```
iot::average
```

```
iot::count
```

```
iot::exclusive_details
```

```
iot::exclusive_times
```

```
iot::inclusive_details
```

```
iot::inclusive_times
```

```
iot::max
```

```
iot::min
```

```
iot::nsysarg
```

```
iot::pathname
```

```
iot::retval
```

```
iot::stddev
```

```
iot::syscallno
```

```
iot::threadAverage
```

```
iot::threadMax
```

```
iot::threadMin
```

```
iot::time
```

Viewing I/O (iot) data in CLI



View in chronological trace order: start_time, time, the rank:thread event occurred in.

openss>>**expview -m start_time,time,id -vtrace**

| Start Time(d:h:m:s) | Exclusive | Event | Call Stack Function (defining location) |
|-------------------------|---------------|-------------------|---|
| I/O Call | Identifier(s) | | |
| Time(ms) | | | |
| 2014/08/17 09:25:44.368 | 0.012356 | 0:140166697882752 | >>>>>>>>>open64 (libpthread-2.17.so) |
| 2014/08/17 09:25:44.368 | 0.027694 | 0:140166697882752 | >>>>>>>>>read (libpthread-2.17.so) |
| 2014/08/17 09:25:44.377 | 0.053832 | 0:140166697882752 | >>>>>>>>close (libpthread-2.17.so) |
| 014/08/17 09:25:44.378 | 0.012347 | 0:140166697882752 | >>>>>>>>>open64 (libpthread-2.17.so) |
| 2014/08/17 09:25:44.378 | 0.007758 | 0:140166697882752 | >>>>>>>>>read (libpthread-2.17.so) |
| 2014/08/17 09:25:44.378 | 0.022821 | 0:140166697882752 | >>>>>>>>close (libpthread-2.17.so) |
| 2014/08/17 09:25:44.378 | 0.037219 | 0:140166697882752 | >>>>>>>>__write (libpthread-2.17.so) |
| 2014/08/17 09:25:44.378 | 0.018545 | 0:140166697882752 | >>>>>>>>__write (libpthread-2.17.so) |
| 2014/08/17 09:25:44.378 | 0.019837 | 0:140166697882752 | >>>>>>>>__write (libpthread-2.17.so) |
| 2014/08/17 09:25:44.379 | 0.035047 | 0:140166697882752 | >>>>>>>>__write (libpthread-2.17.so) |
| ... | | | |
| ... | | | |

Viewing I/O (iot) data in CLI



View the load balance (max, min, and average values) across all ranks, threads, or processes.

openss>>**expview -m loadbalance**

| Max I/O Rank | Min I/O Rank | Average | Function (defining location) |
|--------------|--------------|-----------|---|
| Call of | Call of | I/O Call | |
| Time Max | Time Min | Time | |
| Across | Across | Across | |
| Ranks(ms) | Ranks(ms) | Ranks(ms) | |
| 1.241909 | 0 | 1.241909 | 0 1.241909 __write (libpthread-2.17.so) |
| 0.076653 | 0 | 0.076653 | 0 0.076653 close (libpthread-2.17.so) |
| 0.035452 | 0 | 0.035452 | 0 0.035452 read (libpthread-2.17.so) |
| 0.024703 | 0 | 0.024703 | 0 0.024703 open64 (libpthread-2.17.so) |

View data for only rank nn, in this case rank 0

openss>>**expview -r 0**

| I/O Call | % of | Number | Function (defining location) |
|----------|-----------|--------|------------------------------|
| Time(ms) | Total | of | |
| Time | Calls | | |
| 1.241909 | 90.077151 | 36 | __write (libpthread-2.17.so) |
| 0.076653 | 5.559734 | 2 | close (libpthread-2.17.so) |
| 0.035452 | 2.571376 | 2 | read (libpthread-2.17.so) |
| 0.024703 | 1.791738 | 2 | open64 (libpthread-2.17.so) |

Viewing I/O (iot) data in CLI



View the top time taking call tree in this application run. iot1 indicates see only one callstack. iot<number> shows “number” of calltrees.

openss>>expview -v calltrees,fullstack iot1

I/O Call % of Number Call Stack Function (defining location)

Time(ms) Total of

Time Calls

 _start (sweep3d.mpi)

 > @ 562 in __libc_start_main (libmonitor.so.0.0.0: main.c,541)

 >> __libc_start_main (libc-2.17.so)

 >>> @ 517 in monitor_main (libmonitor.so.0.0.0: main.c,492)

 >>>> 0x4026a2

 >>>>> @ 185 in MAIN__ (sweep3d.mpi: driver.f,1)

 >>>>>> @ 41 in inner_auto_ (sweep3d.mpi: inner_auto.f,2)

 >>>>>>> @ 128 in inner_ (sweep3d.mpi: inner.f,2)

 >>>>>>>> _gfortran_st_write_done (libgfortran.so.3.0.0)

 >>>>>>>>> 0x30fb8db56f

 >>>>>>>>>> 0x30fb8e65af

 >>>>>>>>>>> 0x30fb8df8c8

0.871600 63.218195 12 >>>>>>>>>>>> __write (libpthread-2.17.so)

Open | SpeedShop CLI output into csv form for spreadsheet use.

❖ Create an experiment database

```
EXE=./a.out
export
OPENSS_HWCSAMP_EVENTS="PAPI_VEC_DP,FP_COMP_OPS_EXE:SSE_FP_PACKED_
DOUBLE"
osshwcsamp "/usr/bin/srun -N 1 -n 1 $EXE "
```

❖ Open the database file and use expview -F csv to create a csv file

```
openss -cli -f a.out-hwcsamp.openss
>expview -F csv > mydata.csv
```

❖ Create csv file using a script method

```
echo "exprestore -f a.out-hwcsamp.openss" > ./cmd_file
echo "expview -F csv > mydata.csv " >> ./cmd_file
echo "exit" >> ./cmd_file
#
# Run openss utility to output CSV rows.
#
openss -batch < ./cmd_file
```

Getting Open|SpeedShop output into csv form for spreadsheet use.

openss -f stream.x-hwcsamp-1.openss

Go to Stats Panel Menu

Select the "Report Export Data (csv)" option

In the Dialog Box provide a meaningful name such as stream.x-hwcsamp-1.csv

File will be saved to a file you specified above

Storing csv information into spreadsheet

- ❖ Open your spreadsheet.
- ❖ Either
 - Select open from pulldown menu
 - import stream.x-hwcsamp-1.csv
 - Cut and paste the contents of stream.x-hwcsamp-1.csv into the spreadsheet
- ❖ Use "Data" operation and then "Text to columns",
- ❖ Select "comma" to separate the columns.
- ❖ Save the spreadsheet file

Plotting the performance info from the spreadsheet (Libre office instructions)

- ❖ Open your spreadsheet.
- ❖ Select Insert, then choose Chart
- ❖ Choose a chart type
- ❖ Choose data range choices (data series in columns)
- ❖ Choose data series (nothing to do here)
- ❖ Chart is created. Views can be changed.



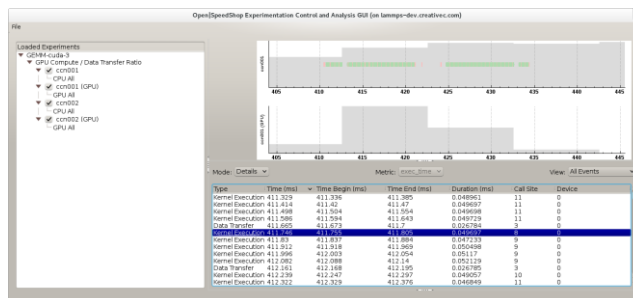
Open | SpeedShop™

Performance with Open | SpeedShop

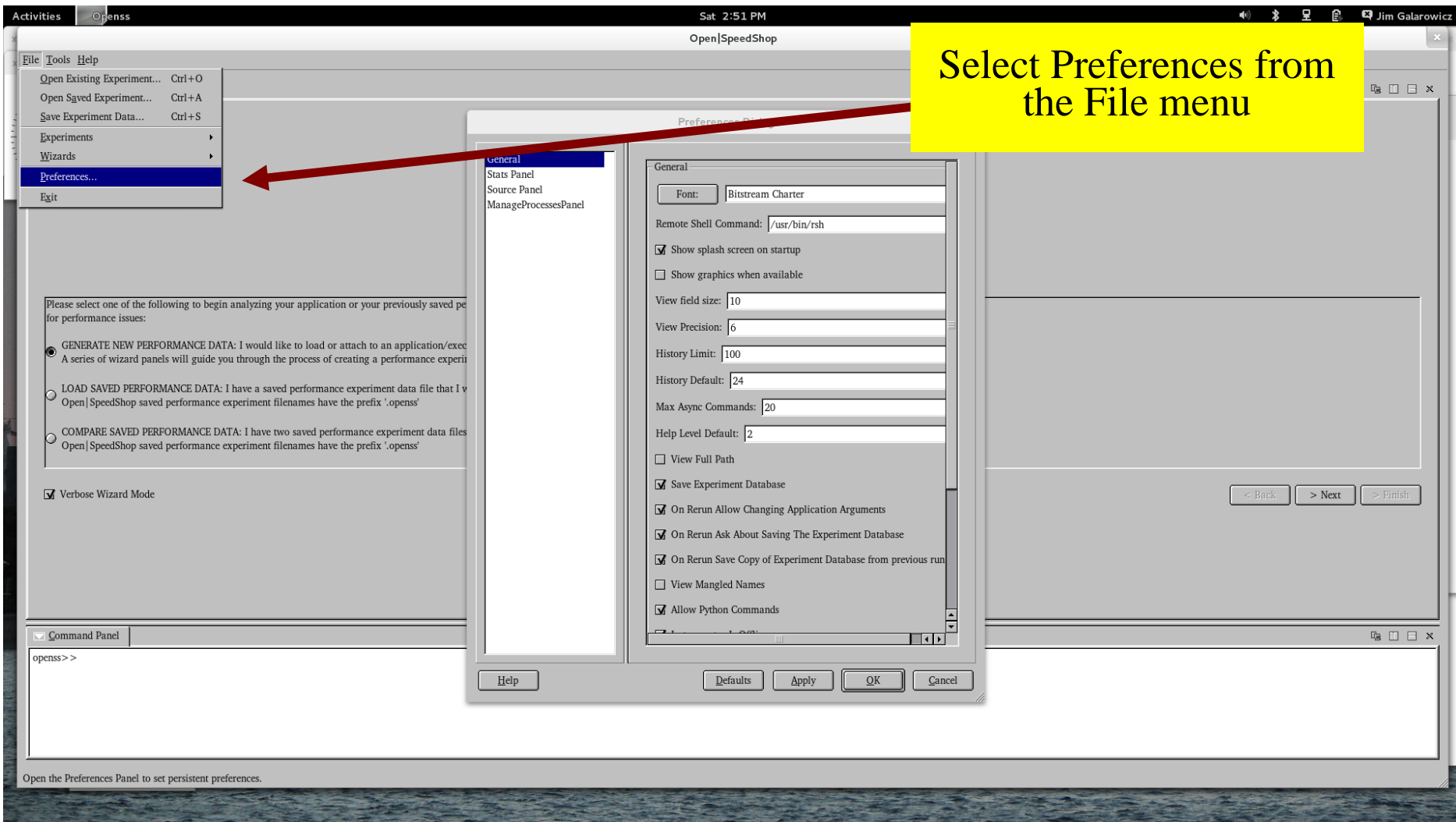
NASA

NASA Open | SpeedShop Update/Training

Supplemental: 2 Preferences



Selecting File->Preferences to customize O|SS



The screenshot shows the Open|SpeedShop application window. The 'File' menu is open, and the 'Preferences...' option is highlighted. A red arrow points from a yellow callout box to this option. The 'Preferences' dialog box is also open, showing the 'General' tab. The dialog box contains various settings such as 'Font', 'Remote Shell Command', 'View field size', 'View Precision', 'History Limit', 'History Default', 'Max Async Commands', 'Help Level Default', 'View Full Path', 'Save Experiment Database', 'On Rerun Allow Changing Application Arguments', 'On Rerun Ask About Saving The Experiment Database', 'On Rerun Save Copy of Experiment Database from previous run', 'View Mangled Names', and 'Allow Python Commands'. The 'Command Panel' at the bottom shows the prompt 'opens>>'. A yellow callout box with the text 'Select Preferences from the File menu' is positioned over the 'File' menu and the 'Preferences...' option.

Select Preferences from the File menu

Enabling the new Command Line Interface (CLI) save/reuse views feature. Looking for friendly user evaluation.

